RasPi

DESIGN
BUILD
CODE

38

Get hands-on with your Raspberry Pi

+ BUILD A **BITCOIN STORE**

CONTROL A DRONE WITH RASPI

Plus Teach Pi to read your emotions

Welcome

This issue sees us take to the skies as we show you how to control your drone with the power of your Raspberry Pi. Currently a headline-making technology, it seems drones are rarely out the mainstream media spotlight. From near misses with passenger jets to saving species from being poached to extinction, the versatile flying gadgets can be put to a myriad of uses from the heroic to the downright nefarious.

Our lead tutorial this issue offers a little taster of a Python module which enables you to program your drone with a predetermined set of flight actions such "fly forward, turn left, turn right and land". Take a swipe to the left and get started now.

Get inspired

Discover the RasPi community's best projects

Expert advice

Got a question? Get in touch and we'll give you a hand

Easy-to-follow guides

Learn to make and code gadgets with Raspberry Pi

Editor

From the makers of

LinuxUser & Developer

Join the conversation at...

@linuxusermag | Linux User & Developer | linuxuser@futurenet.com

Contents

Use the Raspberry Pi to take control of your drone

Create code which enables you to customise and automate drone flights, retrieve flight data and respond to the drone tag

THE PROJECT ESSENTIALS

Raspberry Pi Drone

Drones are becoming ever more popular and mainstream. You may already be aware that Amazon is currently working on using drones to deliver packages directly to your house within hours of you placing an order, creating a new requirement for them to be autonomous and programmable.

This tutorial offers a little taster of a Python module which enables you to write and deploy programs to your drone. The API used is now a few years old (2014) and supports operation with Python 2.7. However, it offers a number of simple methods for beginners to create their own programs and spark your curiosity.

The tutorial begins with a quick walkthrough on installing the required software and libraries. Then jump straight in and create a simple but inspiring program which, when run, automatically launches the drone and then lands it. The tutorial covers the instructions to connect to the drone via the Raspberry Pi's Wi-Fi and then deploy your programs. You will then learn how to automate the drone and program a predetermined set of flight actions; for example, fly forward, turn left, turn right and then land. You

can experiment with the various movements and create your own versions and flight plans. The final section of the tutorial introduces the use of the 'tag' symbol which can be recognised by the drone's front-facing camera. Once recognised, flight data is sent back to the drone and you can use these values to trigger an action or response such

Left The PS-Drone API is used to control the drone from the Raspberry Pi

as flying forward towards the tag or landing the drone.

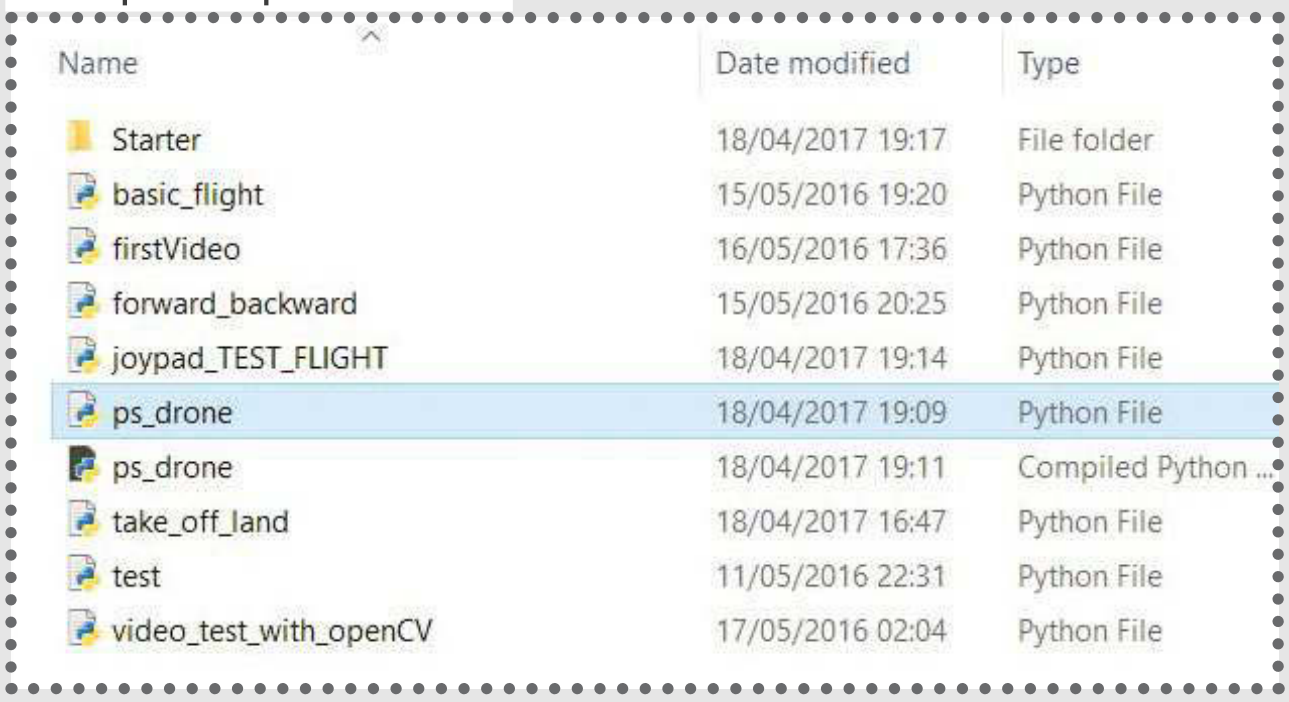
01 Update Pi and install drone API

This project uses a Python API which enables you to access and control the drone directly with Python code. To get started, update your Raspberry Pi using the standard update and upgrade commands (sudo apt-get update, sudo apt-get upgrade). Then, using the web browser on your Pi or other computer, head over to **www.playsheep.de/drone/downloads.html**, right-click on the PS-Drone API program and save it. (The API file is also available in the FileSilo.)

02 Test program – part 1

The PS-Drone API file requires no installation, but it must be placed into the same folder as the Python files that you create to control your drone. Open the LXTerminal and type sudo idle to open the Python 2 programming environment. Begin by creating a simple test program which will launch the drone and then land it. Start a new file and import the time and PS-Drone module.

```
import time
import ps_drone
```



03 Test Program part 2

The next step is to initialise the drone, setting up and opening a communication connection between it and the Raspberry Pi. Begin by initialising the API, then connect to the drone via the software, starting the subprocesses. Next, add the command to launch the drone; this uses the code drone.takeoff().

```
drone = ps_drone.Drone()
drone.startup()
drone.takeoff()
```

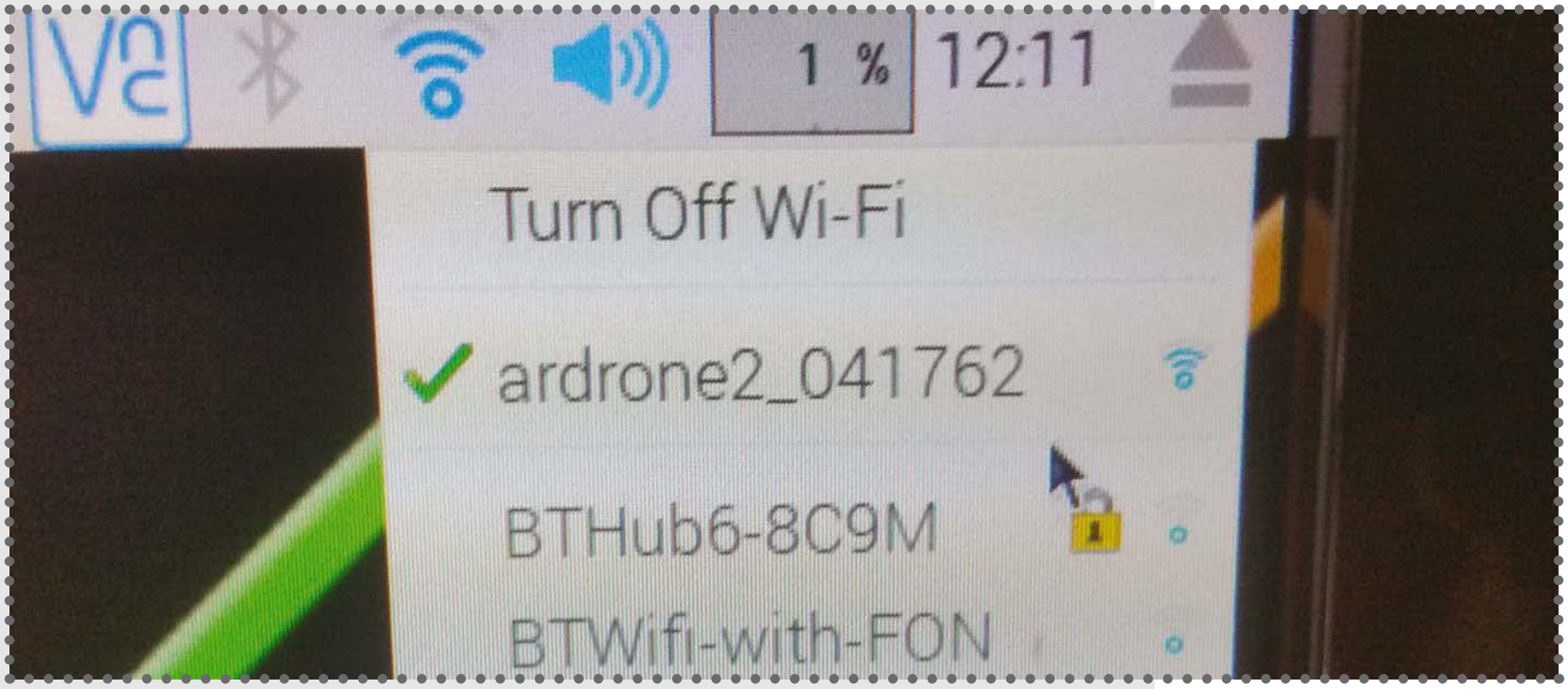
04 Test program – part 3

The last stage is to use the time library to add a short delay between the drone taking off and then landing. This can be used to check the connections are working by setting the delay to two seconds, which is enough time to start the rotors and stop them without the drone leaving the ground. To land the drone, use drone.land(). Add the following two lines to your program and save it, ensuring that the file is in the same folder as the PS-Drone API file.

```
time.sleep(2)
drone.land()
```

05 Transmit a signal

Before the program will interact with the drone, you need to connect to it via the wireless network that it creates. Each drone has an on-board router which creates and broadcasts an open network. Power up the drone and wait for it to run the preflight checks. Success is usually indicated



by four green lights. Then use the network finder on your Pi to search for the drone. Double-click to connect. Note that it requires no key or password to connect.

06 Run the program

To run the program, you'll need to execute it from the LXTerminal window. Open the Terminal and use cd to navigate to the folder where your test program is saved. To run the program, type **sudo python name_of_the_file.py** and press **Enter**. (In this tutorial, the file is named **take_off_land**). Ensure you have enough space to safely launch the drone. If not, then set the time value to one.

```
sudo python name_of_the_file.py
```

```
pi@Dragon-Fire-Rising:~ $ cd Drone-Hacks/3_PS_Drone/
pi@Dragon-Fire-Rising:~/Drone-Hacks/3_PS_Drone $ sudo python land_it.py
```

07 Automate the drone

Now that you have a working connection and program, adapt it to move or fly the drone. Start a new

Python file and import the ps_drone. Then, as before, initialise the API and connect to the drone. Set the drone to take off, but this time set the time delay to 7.5 seconds. This provides sufficient time for the drone to take off, stabilise itself and wait for the next commands.

```
import time
import ps_drone
drone = ps_drone.Drone()
drone.startup()
drone.takeoff()
time.sleep(7.5)
```

08 Fly forward

Add the code to fly the drone forward; this is simply drone.moveForward(). Then add a pause which represents the duration that you want the drone to fly forward for. In this example, the drone flies forward for two seconds. Next, stop the drone. Like a car, this requires a stopping time, so add a short time delay; you can match the delay value used when flying forward. You can test this program before moving onto the next steps.

```
drone.moveForward()
time.sleep(2)
drone.stop()
time.sleep(2)
```

09 Fly backwards

Once the drone has stopped, it will hover until it receives another command. On the next line down, add the code to fly the drone backwards. Again, you will need to add a short time delay, then stop the drone. Finally, set the code to land the drone.

Hints and tips

When connecting to the drone via the Raspberry Pi, there are possible errors. Many have been resolved in the PS-Drone API version two.

1) Sometimes, running a program after a previous program has finished can result in the error 'address in use'. This is because the drone/Pi has not disconnected from the network socket. Simply reset your Pi.

2) Always ensure that the drone has completed its startup checks and is online, indicated by four green LEDs, before you attempt to connect to it.

```
drone.moveBackward()
time.sleep(1.5)
drone.stop()
time.sleep(2)
drone.land()
```

10 Run the program

Ensure that your Pi is connected to the drone's network, as shown in Step 5. Remember that the program needs to be executed from the Terminal window. As before, open it and use cd to navigate to the folder where the program is saved. To run the program, type **sudo python name_of_the_file.py** and press **Enter**.

11 Turning left or right

Modify the same program to alter the flight direction of the drone. This uses the code lines, **drone.turnLeft()** and **drone.turnRight()**. After each call, remember to state the duration or time that the action runs for. For example, using **time.sleep(2)** will turn the drone left or right for two seconds. After the required time, stop the turning action using the code **drone.stop()**. This returns the drone to the hovering state. Add the relevant code to your program and then save it. Connect and run as previously demonstrated in Steps 5 and 6.

```
drone.turnLeft()
time.sleep(2)
drone.stop()
time.sleep(2)
```

Hints and tips (cont)

3) When using the 'tag', it is easier to recognise if the background is in contrast; for example, white paper against a white shirt does not aid identification. Check out the tag hack in action at **https://youtu.be/gXKdUhz7zAw**.

12 Tag detection

The drone software has the capability to use the forward-facing camera to identify and read a 'tag'. (Downloadable from the website or available in the FileSilo.) Once detected, this can then be used to trigger an event such as landing the drone or flying it towards the tag. Begin the program by starting a new Python file; import the time and sys modules. Next, import ps_drone. As before, add the startup and connect to the subprocesses.

```
import time, sys
import ps_drone
drone = ps_drone.Drone()
drone.startup()
```

2_forward_backward.py - F:\Freelance Work\Linux Tutorials\Drone\Drone-Hacks\2_forward_backward.py (3.6.0)

File Edit Format Run Options Window Help

```
import time
import ps_drone                 # Imports the PS-Drone-API

drone = ps_drone.Drone()        # Initializes the PS-Drone-API
drone.startup()                 # Connects to the drone and starts subpr

drone.takeoff()                 # Drone starts
time.sleep(7.5)                 # Gives the drone time to start

drone.moveForward()             # Drone flies forward...
time.sleep(2)                   # ... for two seconds
drone.stop()                    # Drone stops...
time.sleep(2)                   # ... needs, like a car, time to stop

drone.moveBackward(0.25)        # Drone flies backward with a quarter sp
time.sleep(1.5)                 # ... for one and a half seconds
drone.stop()                    # Drone stops
time.sleep(2)

drone.land()                    # Drone lands
```

13 Tag settings – part 1

There are four configuration settings to set. First, reset the drone (line one) so that the status is set to 'good', as indicated by four green LEDs. Use the code drone.useDemoMode(True) to use a dataset of 15 when transferring the data from the camera. Now set which packets will be decoded. Finally, set a small time delay to enable the drone to fully awake after the reset.

```
drone.reset()
drone.useDemoMode(True)
drone.getNDpackage(["demo","vision_detect"])
time.sleep(0.5)
```

14 Tag settings – part 2

In this second set of configurations, start by enabling the universal detection by setting the detect type to a value of 3. This triggers the drone to look for the specific tag. Since you are using the front camera only, disable detection from the ground camera. Then set the drone configuration count.

```
drone.setConfig("detect:detect_type", "3")
drone.setConfig("detect:detections_select_h","128")
drone.setConfig("detect:detections_select_v","0")
CDC = drone.ConfigDataCount
while CDC == drone.ConfigDataCount:
time.sleep(0.01)
```

15 Takeoff and taking a reading

The drone is now configured to recognise and read the tag; add a small time delay, line one. This is useful if you need to move the drone outside before it launches. Add lines two

Hack other drones

You may have a different model or make of drone and there will be compatibility issues with the interfacing. However, a quick search of the internet or GitHub throws up a wide range of software for other makes and models. For example, this project page website **(http://bit.ly/DroneNav)** supports the Parrot BeBop drone and mini copter drones. It covers some exciting hacks such as image recognition, following a particular colour and streaming video.

and three to launch the drone; once deployed, it will continue to hover. Next, create a loop to continually check for the tag and take the readings. On line nine, tagNum returns the number of tags found; tagX returns the horizontal position of Drone in relation to the tag. The vertical position of the drone is collected with the code on line 11 and stored in a variable called tagY. The distance of the drone from the tag, tagZ, is on the penultimate line and orientation of the drone is stored in tagRot. Remember to include the indentations when adding the lines of code.

```
  time.sleep(10)

  ###take off###
  drone.takeoff()
  time.sleep(7.5)

  # Get detections
  stop = False
  while not stop:
      NDC = drone.NavDataCount
  while NDC == drone.NavDataCount:
time.sleep(0.01)
  if drone.getKey():
  stop = True
  # Loop ends when key was pressed
  tagNum = drone.NavData[“vision_detect”][0]
  tagX =   drone.NavData[“vision_detect”][2]
  tagY =   drone.NavData[“vision_detect”][3]
  tagZ =   drone.NavData[“vision_detect”][6]
  tagRot = drone.NavData[“vision_detect”][7]
```

16 Responding to the data

Now set up a conditional, an if statement to display the data and take action. Line three prints out the collected data; note that it is converted into a string as the original data format is returned as a float, ie a decimal. Create a new variable called distance to store the physical measurement of the distance of the drone from the tag. This is stored as an integer, line four. Check if this distance is greater than 300, line six; if it is then trigger an event. For example, set the drone to fly towards the tag for two seconds, lines seven and eight.

```
  if tagNum:
  for i in range (0,tagNum):
  print “Tag no “+str(i)+” : X= “+str(tagX[i])+”
Y= “+str(tagY[i])+”  Dist= “+str(tagZ[i])+”
Orientation= “+str(tagRot[i])
```

Left If the distance to the tag is greater than 300, the drone keeps flying forward

```
drone.takeoff()                 # Drone starts
time.sleep(7.5)                 # Gives the drone time to start

# Get detections
stop = False
while not stop:
    NDC = drone.NavDataCount
    while NDC == drone.NavDataCount:    time.sleep(0.01)
    if drone.getKey():                  stop = True
    # Loop ends when key was pressed
    tagNum = drone.NavData["vision_detect"][0]              # Number of found tags
    tagX =   drone.NavData["vision_detect"][2]              # Horizontal position(s)
    tagY =   drone.NavData["vision_detect"][3]              # Vertical position(s)
    tagZ =   drone.NavData["vision_detect"][6]              # Distance(s)
    tagRot = drone.NavData["vision_detect"][7]              # Orientation(s)

    # Show detections
    if tagNum:
        for i in range (0,tagNum):
            print "Tag no "+str(i)+" : X= "+str(tagX[i])+"  Y= "+str(tagY[i])+"  Dist= "+str(tagZ[i])+"  Orientation= "+str(tagRot[i])
            print (tagZ)

            distance = int(tagZ[i])
            print (distance)

            ### convert to a number

            print (type(distance))
            if distance > 300:
                print ("Moving Forward")
                drone.moveForward()
                time.sleep(2)
                drone.stop()
                #time.sleep(1)
            else:
                print ("close enough")
                #drone.stop()
                #time.sleep(0.5)
```

```
  distance = int(tagZ[i])
  print (distance)
  if distance > 300:
  print (“Moving Forward”)
  drone.moveForward()
  time.sleep(2)
```

17 Close enough

Once the drone is close enough to the tag, (in this program less than 300), then it stops flying forward, line one. Add a short time delay to allow it to stop, line two. Add the two other conditions; if the drone is already less than a distance of 300 from the tag then print ‘close enough’. Finally, respond if the tag is not detected, lines five and six. Save your program code and connect to the drone. (Ensure that the indentation levels are correct; if not, this will cause errors when you run the code.) Execute the program as before, following the method described in Steps 5 and 6.

```
  drone.stop()
  #time.sleep(2)
  else:
  print (“close enough”)
  else:
  print “No tag detected”
  #drone.stop()
```

This concludes a basic overview of getting started with the drone hacks. Now get experimenting!

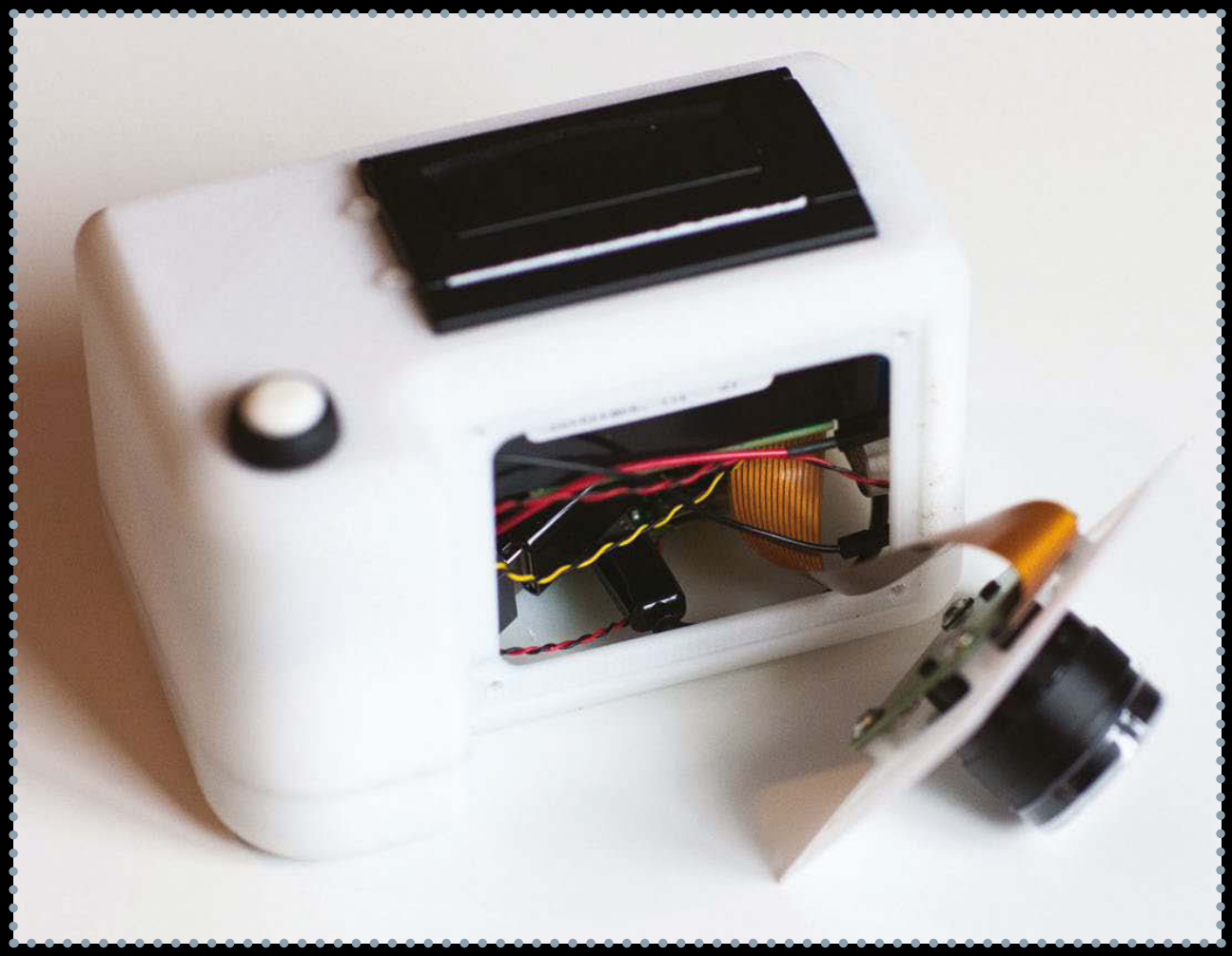
PolaPi Zero

Pierre Muth's camera project is a creative mix of nostalgic Polaroid printing with digital capabilities

How did you come up with the idea behind designing the original PolaPi and the PolaPi Zero?

I was first inspired by a couple of other camera-based development projects, named the PrintSnap and Polatherm Camera. These gave me some ideas for when I developed the original PolaPi model. I like the idea of the Polaroid camera being able to not only show the picture you just took to your friends, but to physically print the picture as well.

After the original version was produced, I sat down with fellow developer Vit Hasek and discussed the possibilities of making the device smaller and more convenient. This is what started the development of the PolaPi Zero. He came with a very clever idea that I didn't have enough time to implement in the first version.



Pierre Muth is a long-time Raspberry Pi tinkerer and engineer, who likes to explore how best to integrate the Pi into everyday items

Would you say it's a blend between classic Polaroid cameras and newer digital cameras?

Yes, I would say so, but there are compromises in both areas. The print quality is obviously bad compared to Polaroid cameras, and the picture files you get are not as great as the ones you can obtain with a recent smartphone. However, in addition to the satisfaction of building something and not only using something, the prints are a lot cheaper and it is fun to use with friends. It opens the door to creative modes and is accessible enough to be rebuilt by a lot of people.

Due to the intricate details of this project, has it been difficult forming a finished product?

I had difficulties on both versions, mainly due to my hardware choices and my lack of knowledge on certain domains. On both versions, the main issue was the internal buffer of the printer. The print speed changes with the darkness of the image. It means if the print is bright, less pixels have to be added and thus the paper advance is quicker, and the other way around. The screen I've chosen also gave me some head-scratching. I was close to choosing a more standard way, such as the PiTFT of the first version. These Sharp 'memory LCDs' are nice to look at, though, so I decided to continue. They are perfectly visible in sunlight as they are reflective LCD and are – as the printer – pure monochrome.

At first, I started by generating the image data to be sent on the SPI port in the program, but I faced some problems when sending the 12 Kbytes of the image in one block. Fortunately, I found the wrobell library for

THE PROJECT ESSENTIALS

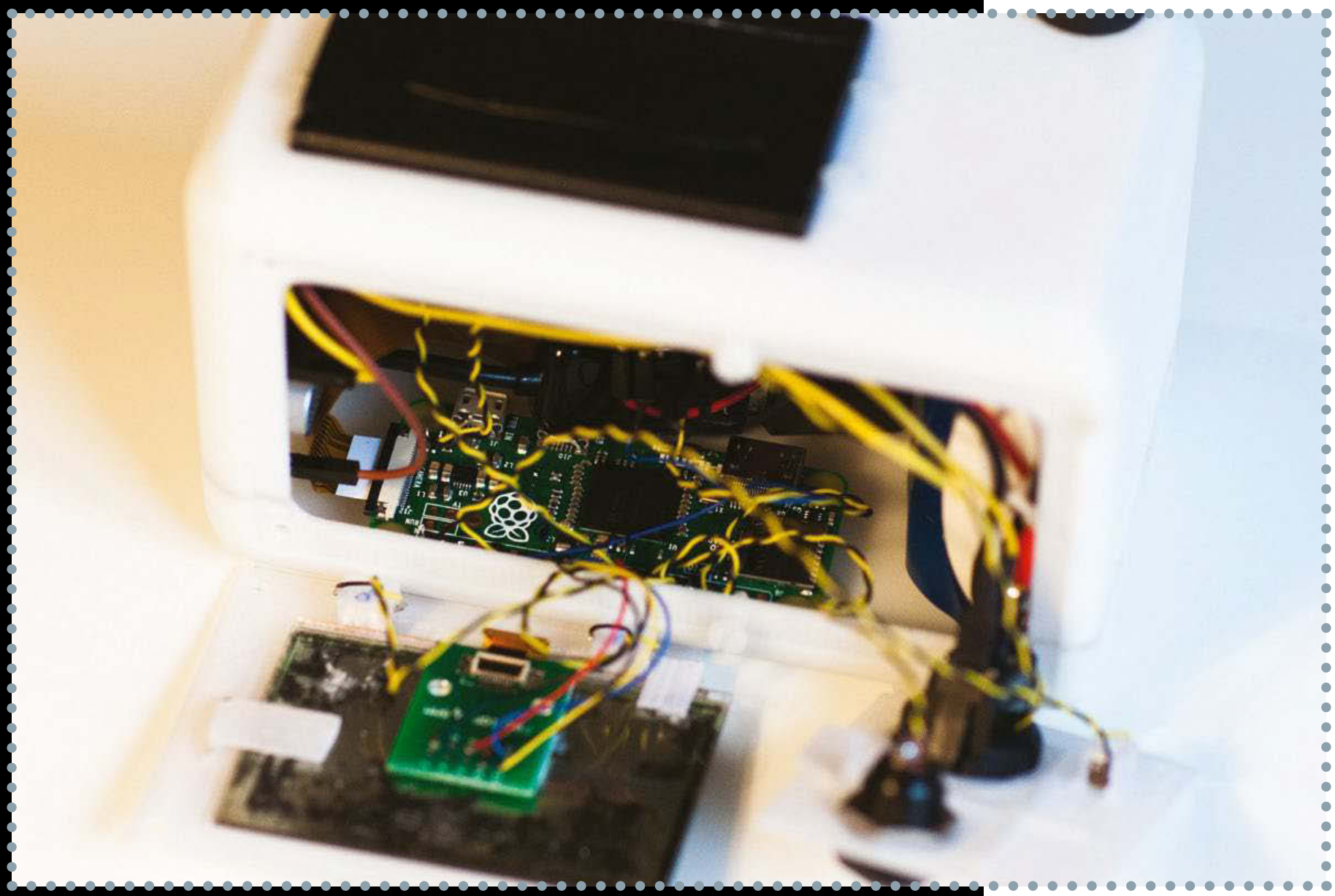
- Raspberry Pi Zero
- 4.3" composite LCD screen
- IRF7319 dual MOSFET
- LM393 voltage comparator
- Lithium charge controller and power board
- Audio amplifier
- Lithium batteries
- FPC connectors and cables
- BAT54C dual diode

this LCD: https://github.com/wrobell/smemlcd. I spent a bit of time figuring out how to compile the library with Autotools and deal with the error messages it returns.

The case itself looks fantastic. Did you use anything in particular to design and print it?

I started from scratch with the Autodesk 123D design free software. If you're a bit familiar with CAD software, it seems quite limited at first. On the other hand, it is possible to achieve a lot of things within an hour, including the learning time.

I tried to limit the number of printed pieces. For tactile buttons, it is a small strip and a square part to hold



the button in place. The front and back plates are fixed with four screws, and the hole threads are just made with a tap in the case plastic. The case can then be easily opened.

What sort of printer did you use for the project? Was it easy to implement into the small frame of the PolaPi Zero?

I think I used one of the cheapest thermal printers out there – you can easily get it from Adafruit. It is the Cashinotech CSN-A3. It is meant to be slot-in, so just four small screws will hold it well in the case. It uses a special kind of paper, the very cheap receipt thermal paper, and you can find much of it from recycling outlets. There is a big debate about the waste of this paper type, as the receipts are usually just thrown away. Nevertheless, here the aim is different. You will never print as many tickets as a supermarket, and in most cases the prints are meant to be kept.

How effective is the Raspberry Pi in this project? Did you find it easy to work with?

Starting with the Raspberry Pi leads you into a vast world of knowledge to acquire, especially if you're not confident with Linux. I started playing with them from their first version and I still play with RPIs because of the enormous community. All the installation process of Raspbian became very easy and the libraries development have aided me in my progress. If you still encounter a problem, there is a large probability that someone else has both faced it and provided a solution.

Like it?

Raspberry Pi camera projects are plentiful, and there are several we highly recommend you check out. One of the best is the Polatherm, which is one of the first to be based on the thermal printing method: **www.dkia.at/en/node/125**.

The Raspberry Pi Zero is a bit more 'closed' than B models, as there is only one USB port. I liked the facility that copies the Wi-Fi network setup file (wpa_supplicant.conf) from the FAT32 partition to the right place at boot time. I then could work on it by connecting a USB Wi-Fi dongle; so, no screen, keyboard or mouse.

After spending some time with it, how do the end photos look? Are you happy with the results?

Do not expect good prints, of course. It is low-resolution (384x640 pixels) and only black or white in its current form. The grey gradients are the result of the error-dithering algorithm, so don't look at them too closely. However, they're unique prints, and a lot of fun to produce. In the future, I'll be looking to improve the quality of them further.

Further reading

Pierre has taken time to upload everything you could possibly want to know about the project over on his Hackaday page. He provides a tutorial of sorts for you to re-create your own PolaPi Zero: **https://hackaday.io/project/19731.**

Make a Bitcoin cold store

Put your Bitcoins on ice and use a Raspberry Pi as a cold wallet to store all your digital currency safely offline

As you're probably aware, the cryptocurrency Bitcoin can be used to make anonymous and irreversible online payments, similar to hard cash. Bitcoin is a huge leap forward for privacy-minded people, but the fact that it's a digital currency means that hackers often target owners of Bitcoin software wallets. In this tutorial, we'll explore a solution using the lightweight Bitcoin Wallet Electrum on the Raspberry Pi. The Pi will host a 'cold' wallet which is kept offline at all times.

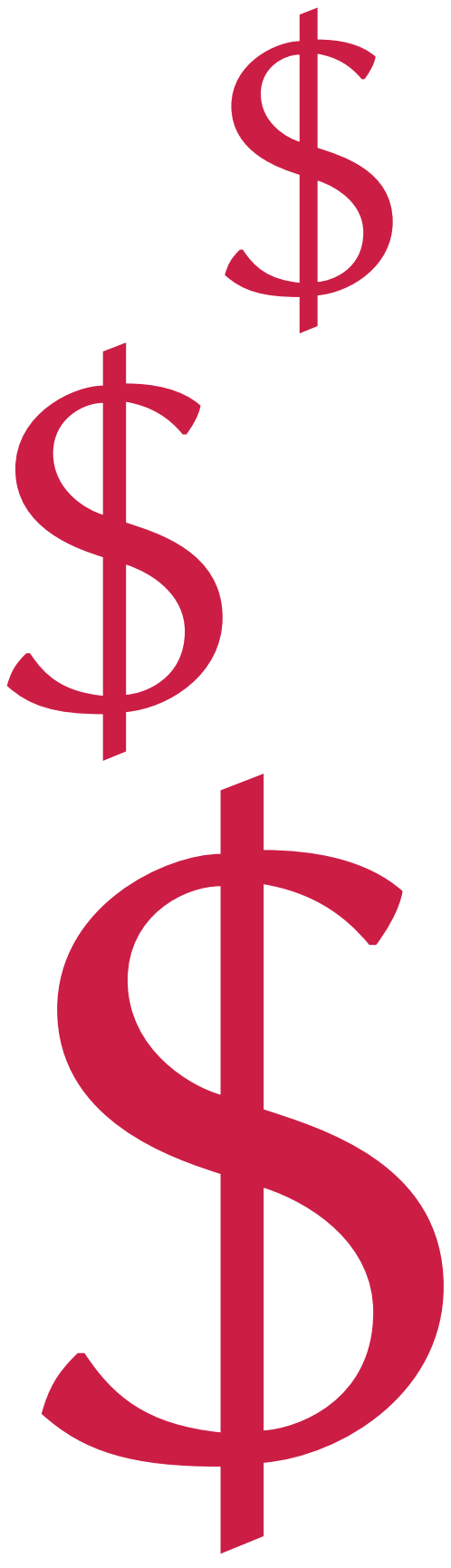
You'll also set up an online 'watching' wallet on another computer so you can see payments as they come in. Payments are only sent by generating a transaction file with your watching wallet, which you transfer manually via a USB stick to your 'cold' wallet to be digitally signed.

As your cold wallet is offline, only people with physical access to it can make payments, which enables you to store Bitcoins safely.

This guide assumes that you are familiar with the basics of Bitcoin. If not, then visit this useful introduction page **http://bit.ly/2m7Ct2X for a rundown.**

THE PROJECT ESSENTIALS

Suitable for all models of Pi (Pi Zero recommended). For security reasons, you need a dedicated Pi for this project.



01 Install software

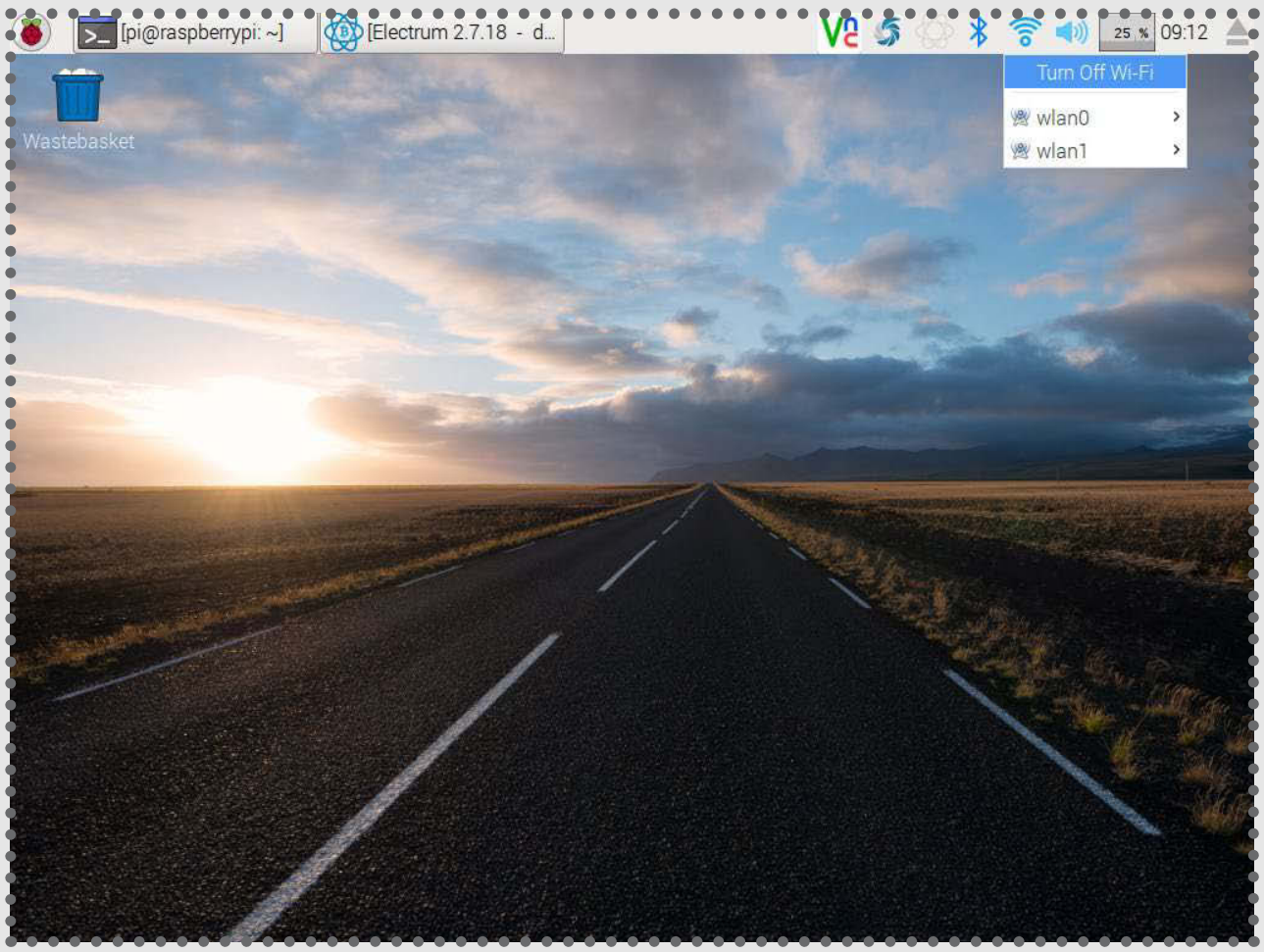
Connect the Pi to a screen and keyboard. Open Terminal and run these two commands:

```
sudo apt-get update
sudo apt-get upgrade
```

For security reasons, you should only use this Pi as a wallet for future purchases.

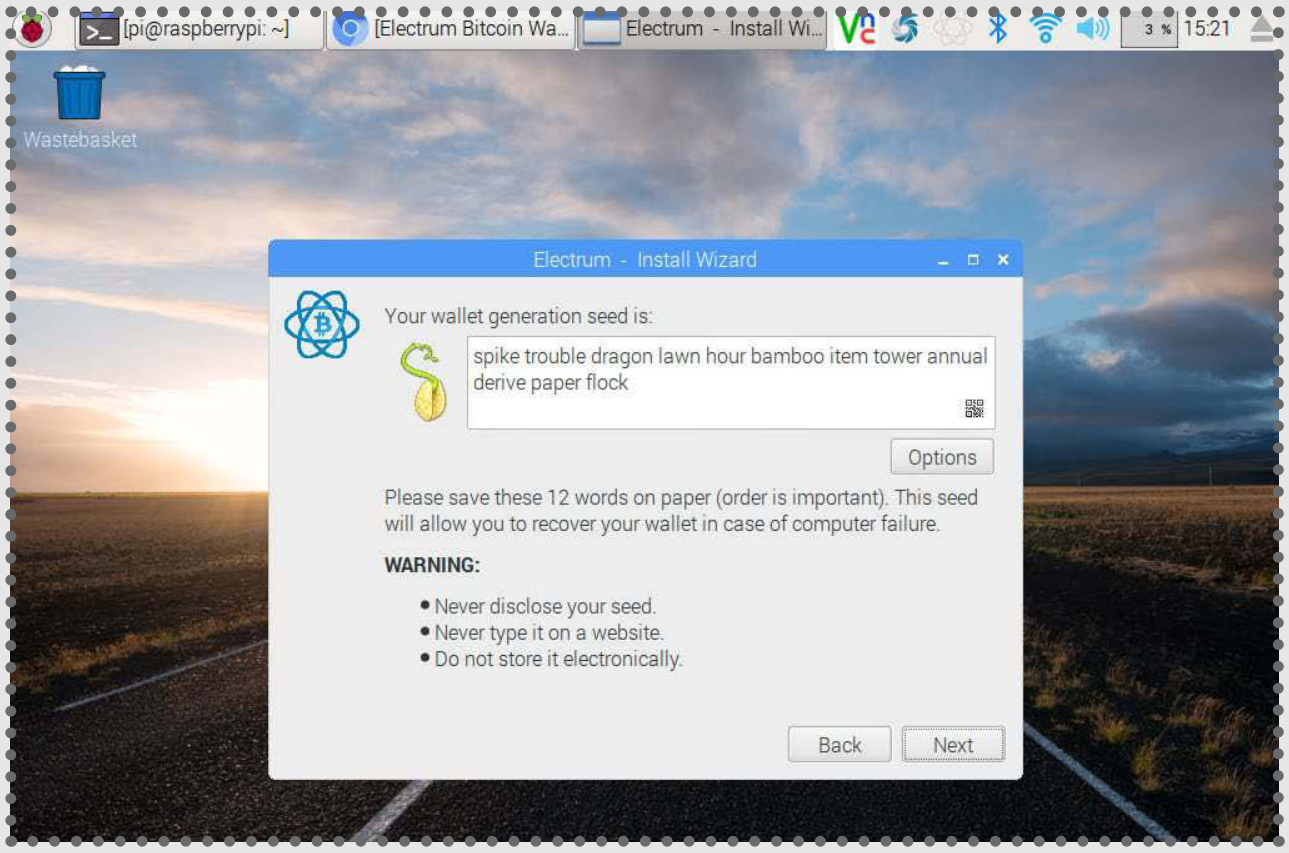
The Raspbian repositories (repos) have an old version of the Electrum wallet, but it is best to download the latest version manually.

Next, run the following two commands to install the latest wallet software with sudo apt-get install python-qt4 then sudo pip install **http://download.electrum.org/2.7.18/Electrum-2.7.18.tar.gz.**



02 Disconnect from the internet

This step is essential for ensuring your wallet is safe. Remove any network cables and/or Wi-Fi adaptors connected to your Pi. If you use a Pi 3, simply click on both the wireless and Bluetooth icons on the Pi and choose 'Turn off Wifi' and 'Turn off Bluetooth' respectively. Keep your Pi connected to a monitor and keyboard, as you'll need this for sending payments later. Once you're offline, run the command electrum in Terminal to start the wallet



03 Generate wallet seed

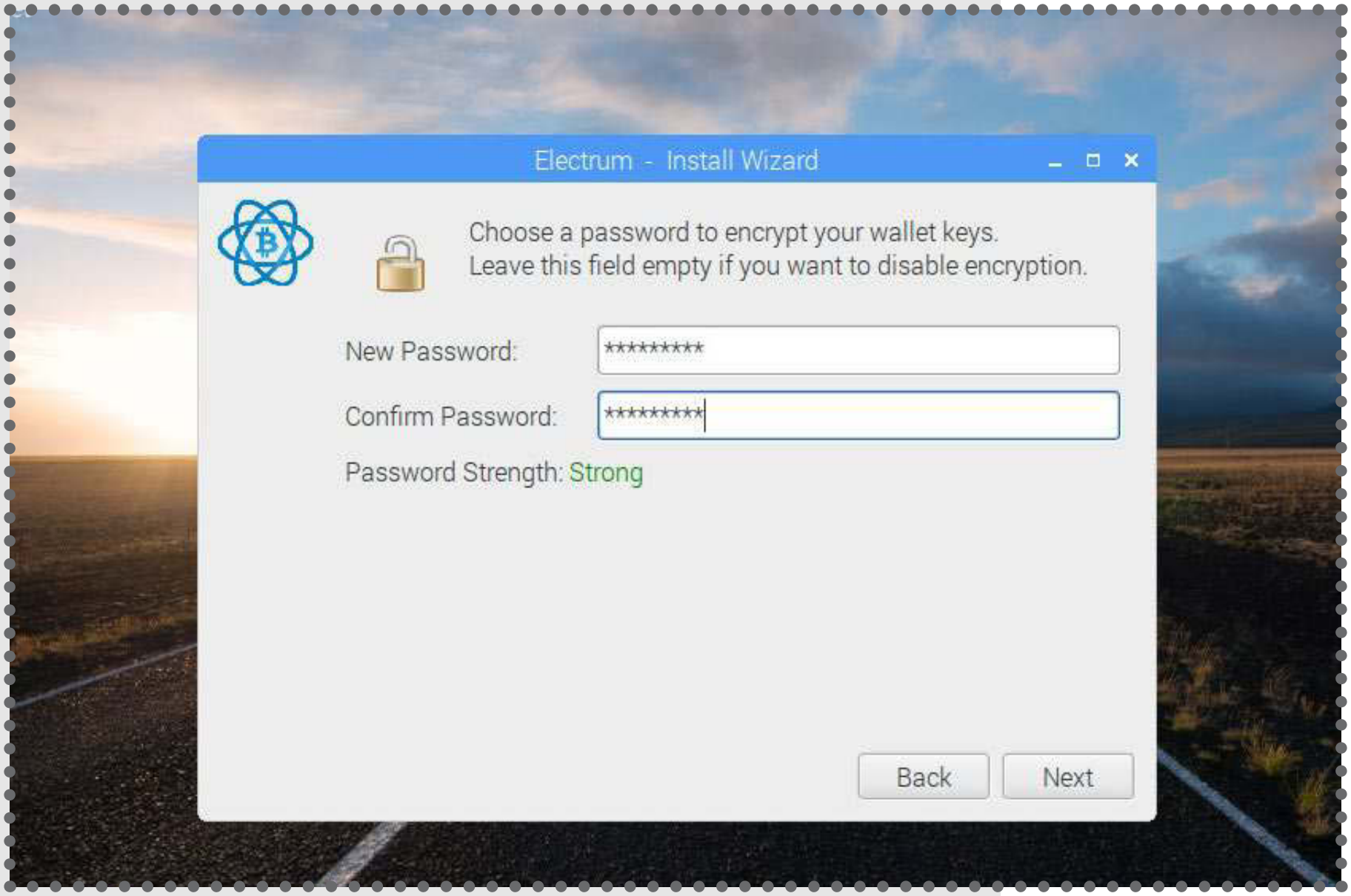
Choose 'Auto Connect' on the first Electrum window, then click 'Next'. On the next screen choose a 'standard' wallet and 'Create a New Seed'. Electrum generates 12 random words for wallet seed and do as the program requests and write these down on paper. These random words will enable you to restore your wallet if anything happens to your Pi. Click 'Next' to continue.

04 Confirm and store wallet seed

Use the words you wrote down earlier to retype your wallet seed. Remember that in the wrong hands, the seed can be used to steal from you, so store the paper in a safe place or better yet memorise the words. Try using the Loci method to memorise the words by associating them with places in a familiar location such as your home.

05 Choose your wallet key password

After you have confirmed the wallet seed you can optionally set a password to protect this copy of the

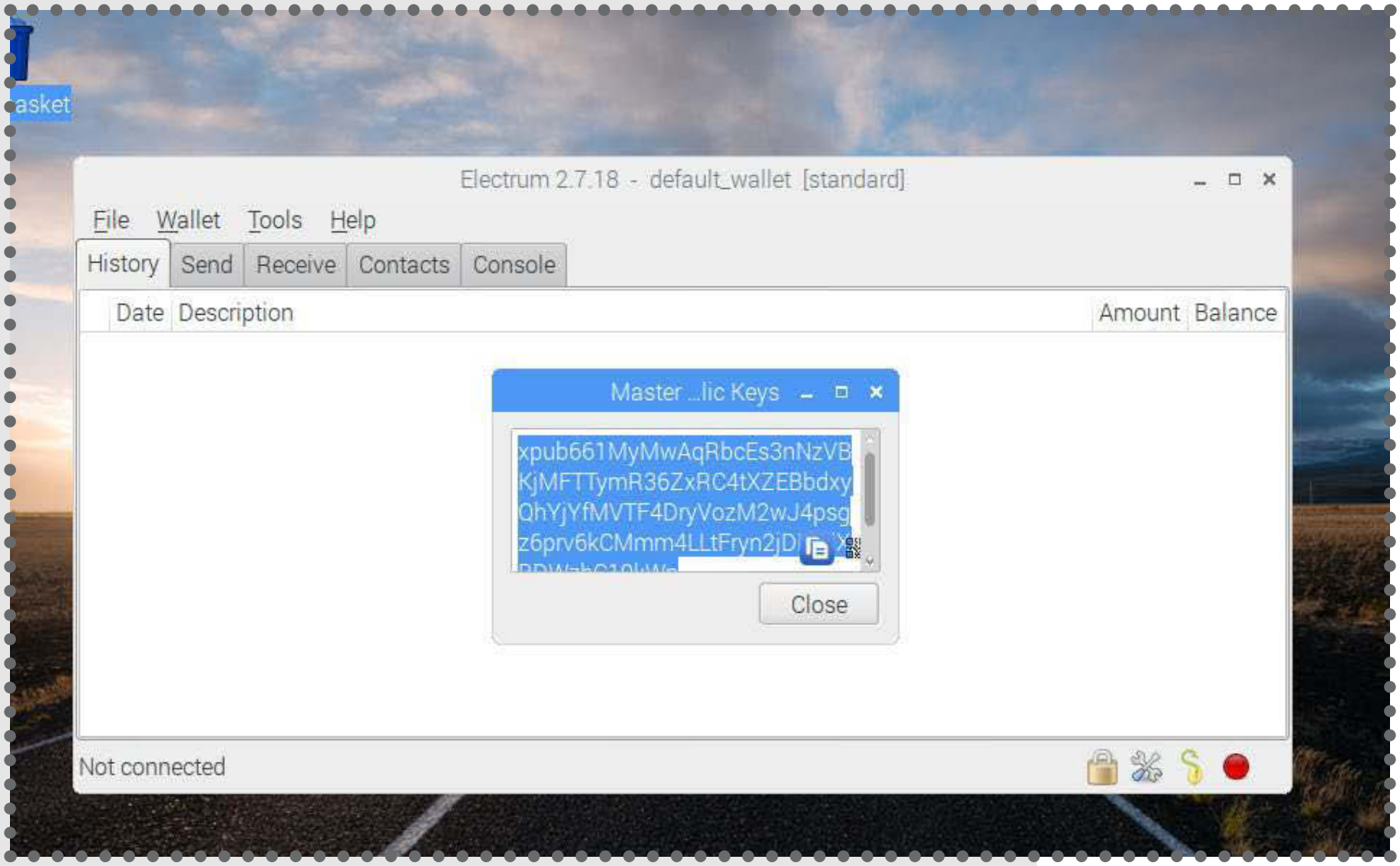


wallet on the Pi. You'll need to re-enter this password when sending Bitcoins. Ideally choose a password that's at least 10 characters long with a mixture of letters, numbers and symbols. Electrum alerts you if a password is weak, moderate or strong. Do not use any of the words in your wallet seed for the password.

06 Export master public keys

In the Electrum Wallet, click the 'Wallet' menu option then 'Master Public Keys'. This will generate a string of text that you'll use on a separate computer for your 'watching wallet' so you can receive payments.

Click the blue icon to copy the key then go to Applications > Accessories > Text Editor to paste it into a text file. Insert a USB stick and save the new text file to it.



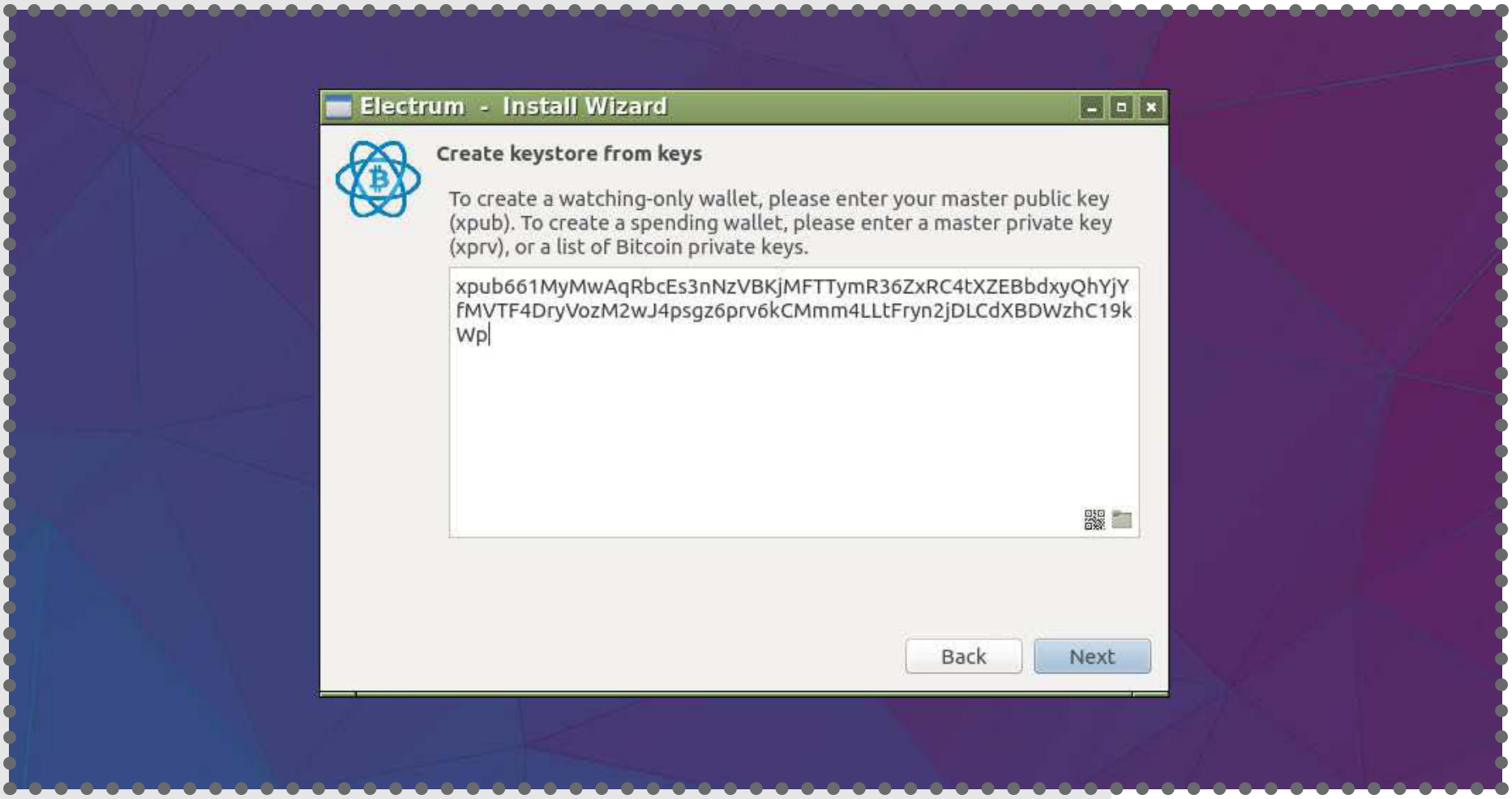
07 Install watching wallet

Safely eject the USB stick from the Pi and insert the stick into the computer that you want to use for your 'Watching Wallet'. Electrum works on most platforms but for security reasons, and because we'd always recommend GNU/Linux over any other platform, place your watching wallet on a spare Linux system, if possible.

If you're using a Debian-based distro like Ubuntu, repeat Step 1 but make sure that you install python-pip as well as python-qt4. For other platforms follow the instructions at **https://electrum.org/#download**, but we'll ignore those and move on.

08 Restore master keys

Launch the Electrum client on your online machine. In Linux, you can simply run electrum from the Terminal. Choose 'Auto Connect' once again on the first screen



and click 'Next'. Click on the 'standard wallet' option once again and then on the 'Keystore' screen choose 'From Public or Private Keys'.

On the next screen, paste in the master key in the text file from your USB stick. Read the warning and click 'OK'.

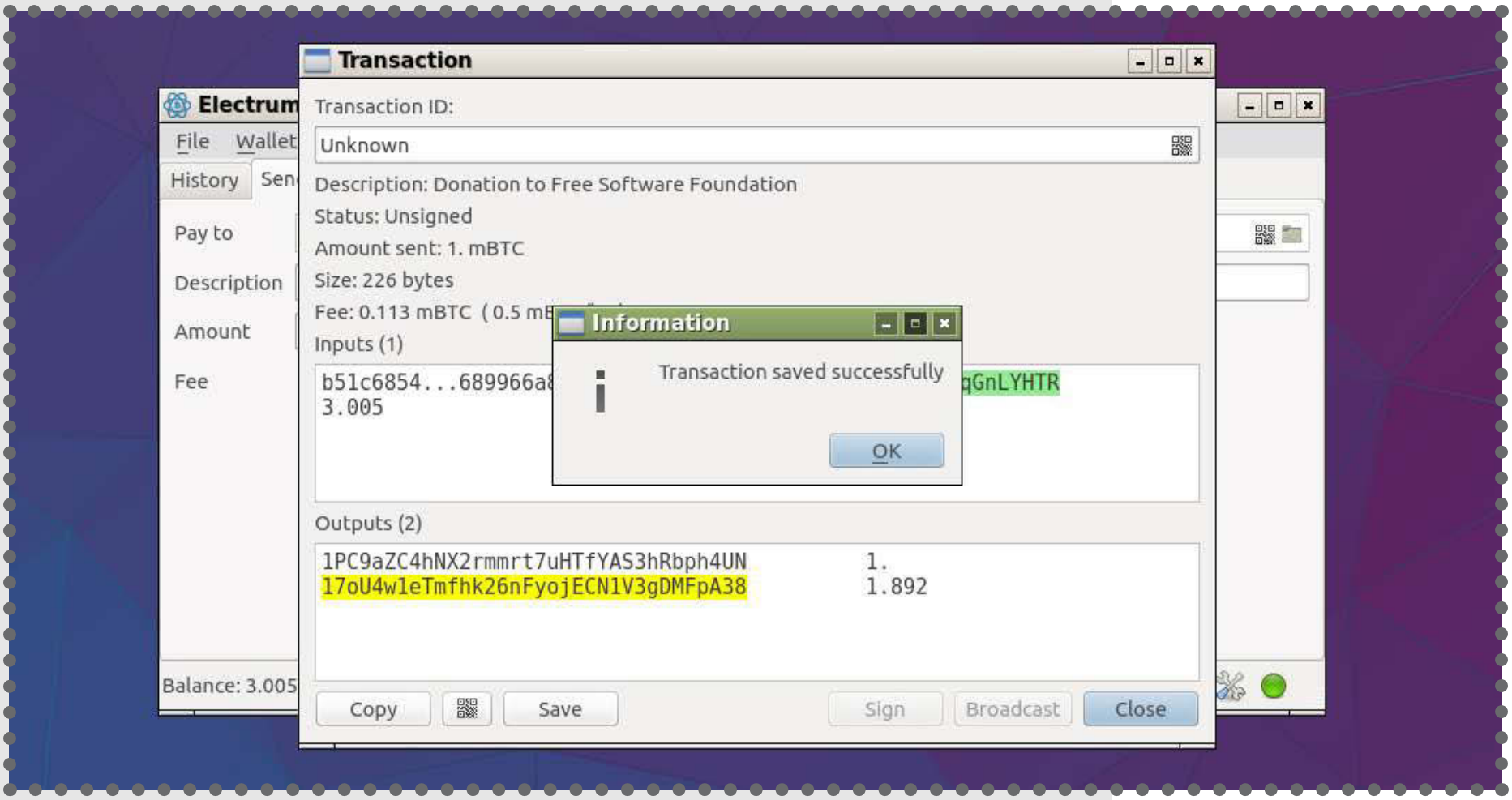
09 Set up addresses

Electrum will now generate addresses for you and when the main window opens click on the 'Receive' tab to view your address. You'll see both a text string and a handy QR code for your new address. Type a description for your receiving address if you wish, then click 'Save'. You can generate more addresses later.

10 Test wallet address

Using your current Bitcoin wallet, try to send some funds to your new wallet address. If possible, use a client which supports scanning QR codes, so there's no danger of getting the address wrong.

Once the payment is sent, the unconfirmed new balance



should show in the bottom-left corner of the Electrum Window. Now, you'll need to wait until the balance is confirmed before proceeding.

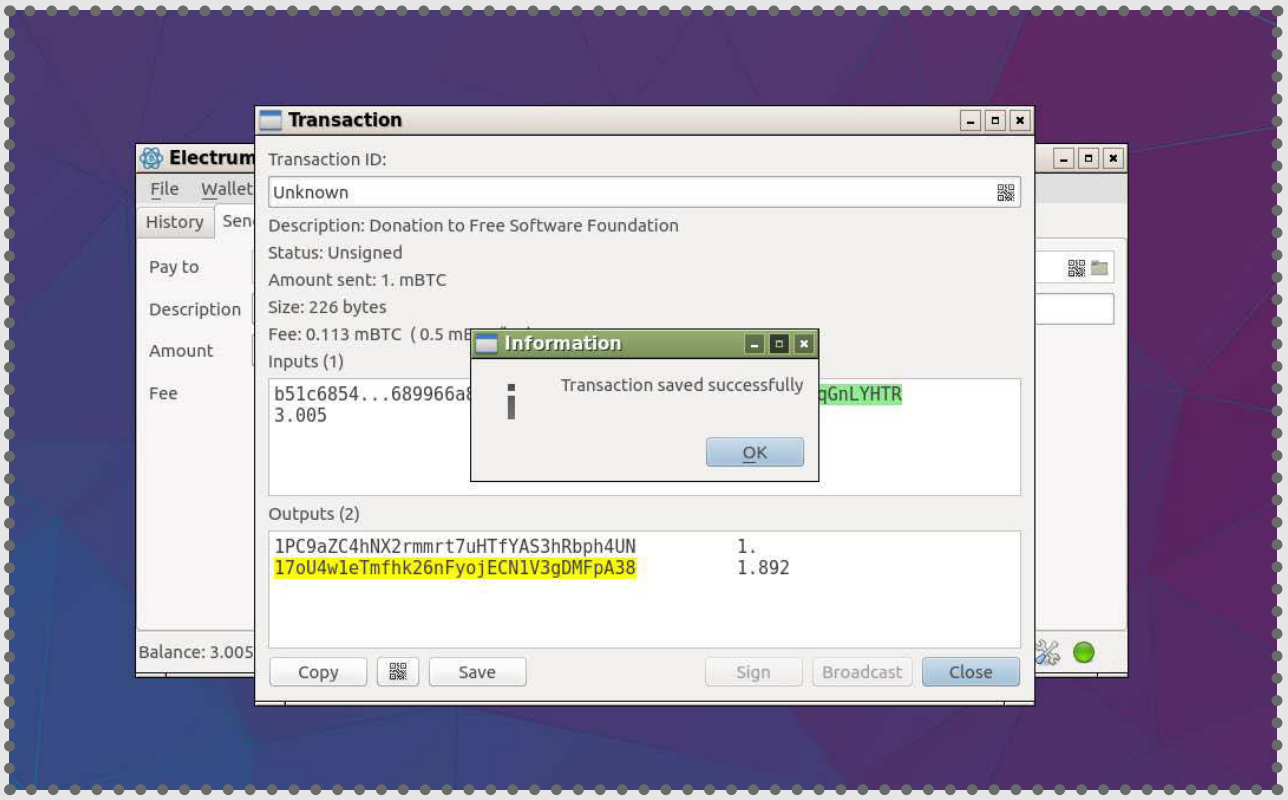
11 Set up sending

Now that you can receive funds, it is time to test if you can send them. Find your recipient's wallet address. For this tutorial, we are using the Bitcoin Address of the Electronic Frontier Foundation (EFF) to send a donation.
Click on the 'Send' tab in Electrum and paste the address into the 'Pay to' field. Add a description if you want to and then click 'Preview'.

12 Save transaction

Click 'Save' in the new window to save the TXN (transaction) file and navigate to your USB stick and place it there. Electrum confirms that the transaction is saved successfully. You can click 'Close' to shut down the window. A transaction needs to be digitally signed by the cold

wallet on your Pi to send the funds. Safely eject the USB stick from your online computer and connect it to the Pi. from your USB stick. Read the warning and click 'OK'.



13 Sign transaction

Open the Electrum client on your Pi and click on the 'Tools Menu'. Select Load Transaction > From File and navigate to the transaction file on the USB stick.

Details of the transaction will load. Check everything is in order then click 'Sign'. Enter your wallet password. Click 'Save' to store the newly signed transaction file to the USB stick. Click 'OK', followed by 'Close' and safely eject the USB stick from the Pi.

14 Broadcast transaction

Next, you'll need to insert the USB stick to your computer with the 'watching' wallet. Open Electrum and click Tools > Load Transaction > From File. Choose the signed file on the USB stick. A new window opens with the transaction details. Double-check these and click 'Broadcast' to send them to the Bitcoin network.

Anonymise your net connection

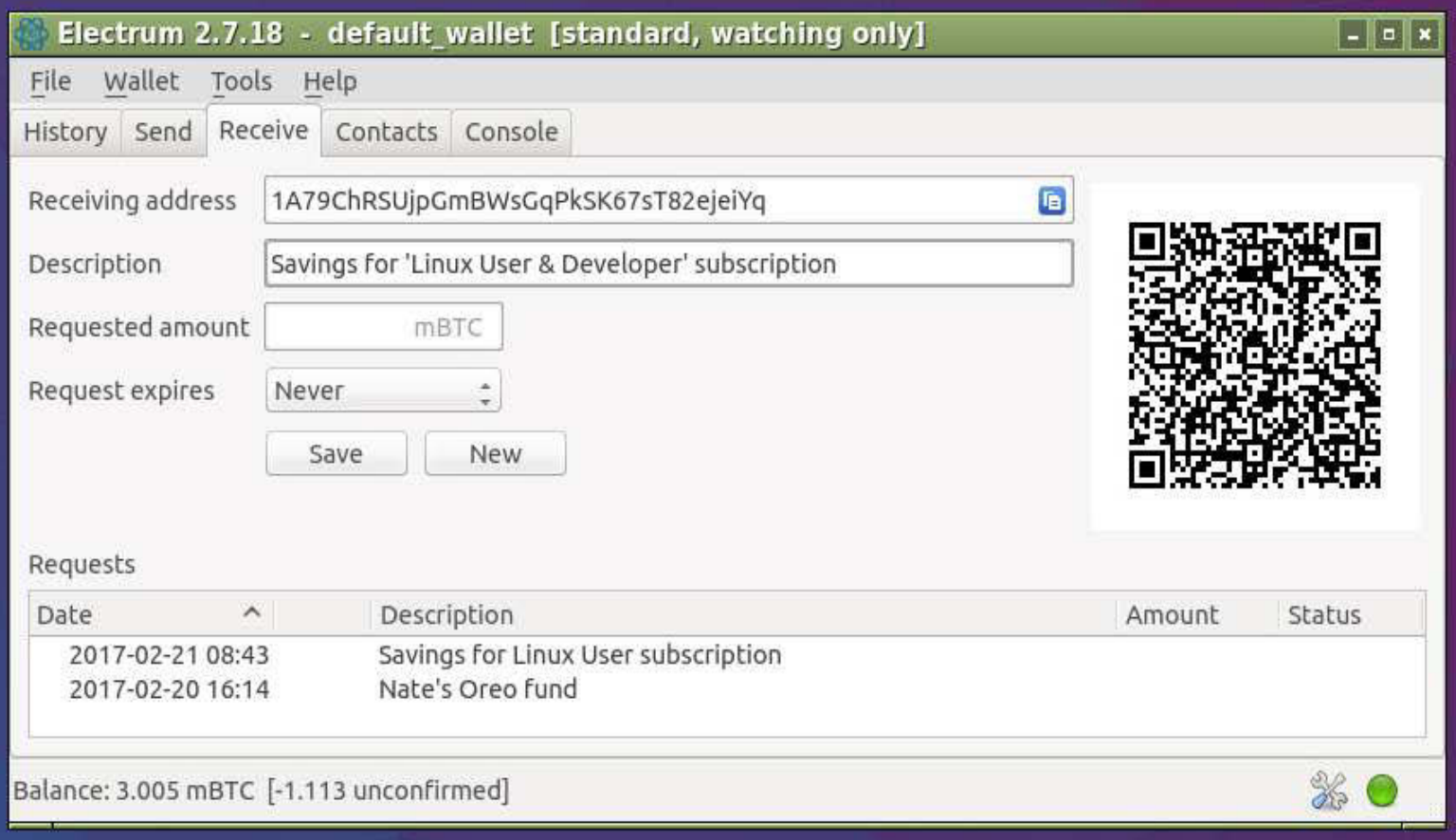
By default, anyone snooping on your connection can see your IP address connecting to the Electrum servers. Your IP could be used to link your identity to specific addresses in your 'watching' wallet. If you have Tor running on your system, you can route Electrum's connection through the Darknet by going to Tools > Network. Select 'SOCKS5' as your 'Proxy'. The rest of the information is filled in already, so just click 'OK'.your needs.

You will see a confirmation message. Click 'Close' to see the unconfirmed amount in the Electrum window. The processing times for a transaction will vary based on the fee that you paid.

15 Generate new addresss

To prevent your Bitcoin transactions from being traced, you should generate a new address for receiving each payment. In the 'Receive' tab of Electrum, click 'New' to create a new one. Enter a description if you like and click 'Save'. You can move between different wallet addresses from the 'Requests' pane. Repeat Steps 10-14 to receive and send funds from this new address.

Having worked you way through this tutorial, you'll now have a 'cold' wallet set up, a method for making payments and an online 'watching' wallet for tracking payments.



Part 1: Command the seas with Sense HAT Battleships

Use the features of the Sense HAT to code the classic strategy game of Battleships complete with sound effects

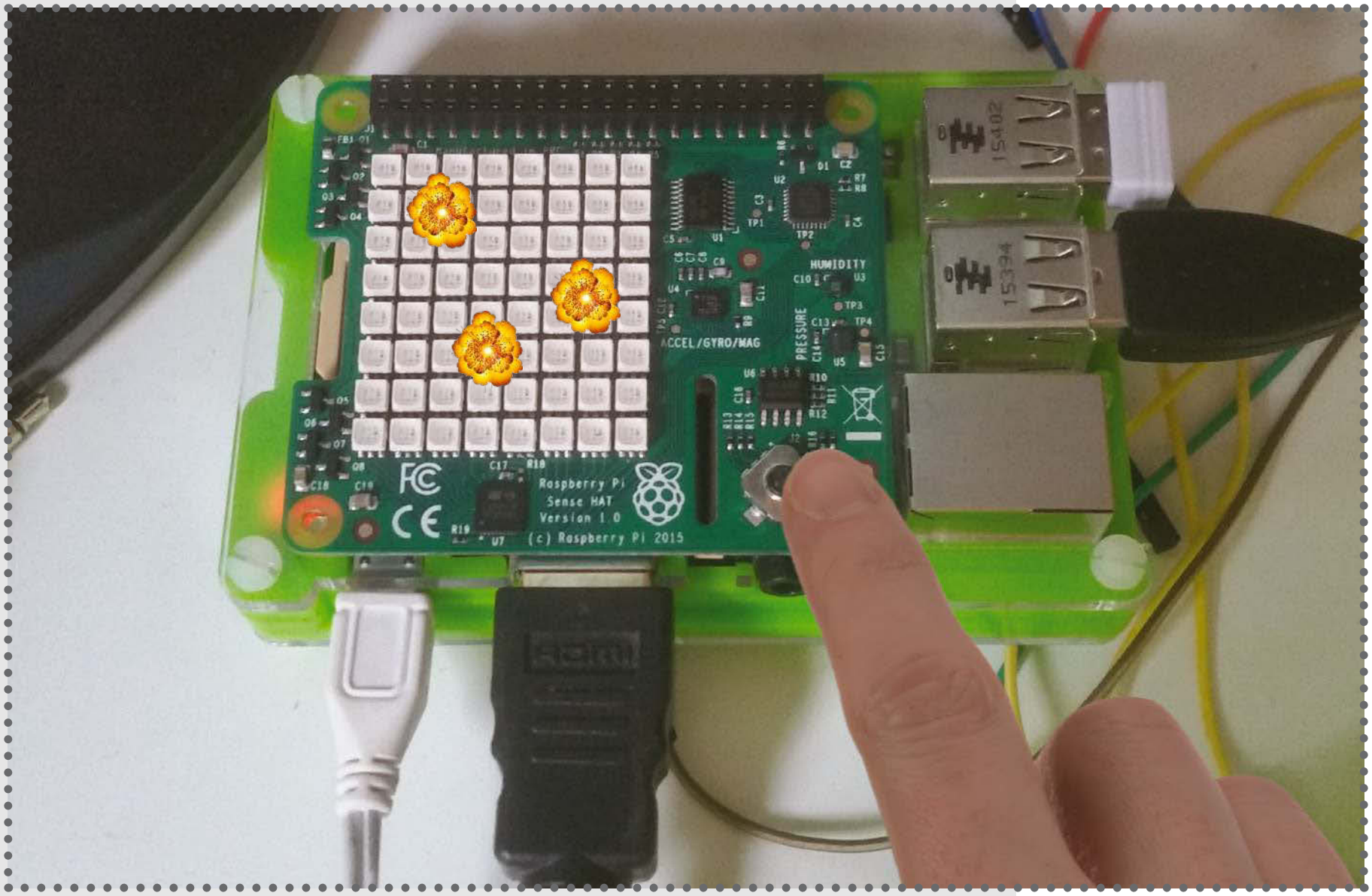
The classic game of Battleships engages you to find and locate your opponent's ships by launching torpedoes to try to sink their convoy. In part one of this two-part tutorial, you will create a version of the classic game for the Raspberry Pi Sense HAT. When completed, your program creates a random layout of enemy boats, some ammo and mostly water. Use the Sense HAT joystick to select one of 64 locations on the sea (LED matrix). Then press **Enter** to fire a torpedo.

The program makes use of two lists: one to store the location of the enemy ships, ammo and water, and the second to track the location of your ship. When you fire a torpedo, your current position data is compared with the first list and calculates if you have missed or hit a ship.

If you miss, you will hear a splash of water; if you hit another ship, you'll hear the explosion. You may even find an ammo dump. As you play, each of the 'LED squares' is colour-coded: blue for water, red for a ship and yellow for ammo. The game frequently updates you with the number of ships you have left to destroy. When

THE PROJECT ESSENTIALS

Wi-Fi enabled Raspberry Pi
Sense HAT



you have used all your torpedoes, or have destroyed all the ships, the game ends and you're updated with your score. You can select to play again and a new random layout is created. Check out the game in action here: **https://youtu.be/SolWA1JHCac.**

01 Import modules and set up – part 1

Open your Python 3 editor and create a new file. Begin by adding the random and time modules. Next, import Pygame and the controls; this enables the Sense HAT joystick and allows you to move and select a single LED position on the matrix. Add the code to import the Sense HAT API.

```
import random
import time
import pygame
from pygame.locals import *
from sense_hat import SenseHat
```

02 Import modules and set up – part 2

Next, initialise Pygame and the Pygame mixer. The mixer enables MP3 files to be played and is used to add sound effects. Set the size of the game window; this is required to enable the joystick to be used within the game. However, it will not display anything, so we set it to a minimal size. Lastly, set the variable sense to SenseHat; in addition, you can reduce the brightness of the LED matrix using the code sense.low_light = True – this will save your eyes!

```
pygame.init()
pygame.mixer.init()
pygame.display.set_mode((200, 200))
sense = SenseHat()
#sense.low_light = True '''enable to turn
  down brightness'''
```

03 Create variables for your ship

Each individual LED on the matrix can be referenced by an x and y value. First, we create the variables to store these two values. This enables you to plot your ship and move it around the grid. Create a variable to store the current score and then set the starting values for the score, the x value and the y value all to zero.

```
global x #led position
```

```
  global y
  global score
  x = 0
  y = 0
  score = 0
```

04 Code the sea

The next few steps build the main gameplay mechanics. Begin by defining a function which stores all the game code. Then import the x, y and score values. Finally, create an empty list called Sea2 which will store the data about the location of the enemy boats, ammo and empty water.

```
  def main():
    global x
    global y
    global score
    Sea2 = []
```

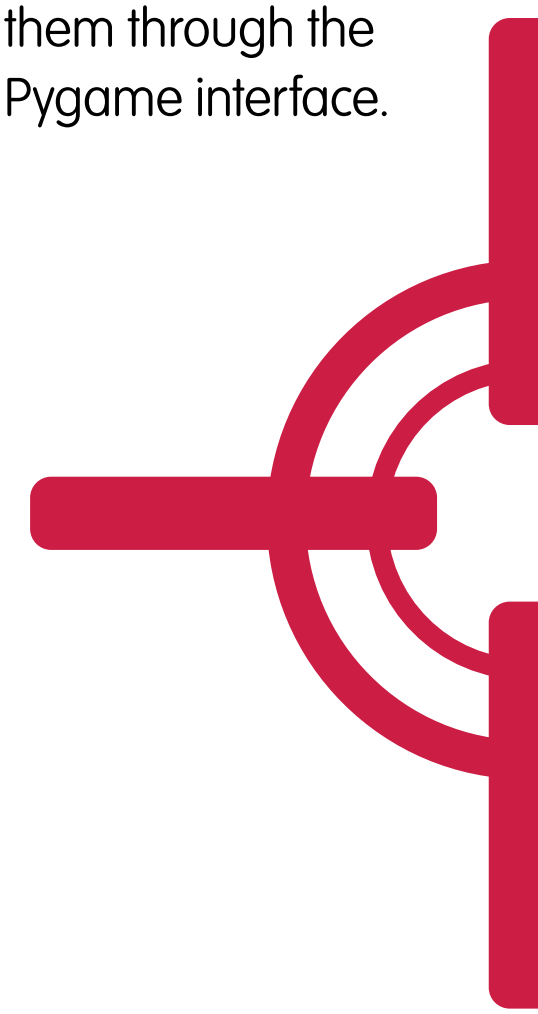
05 Choose ship colours

Select and assign the colours for the water, ships, ammo and the colour for an enemy ship that has been hit and destroyed. Each colour consists of a combined RGB value between 0 and 255. The colours used in this tutorial are: red for enemy ships, blue for water and yellow for ammo.

```
  '''colours for ships and water'''
    ship = [160, 200, 140]
    water = [0, 0, 255]
    hit = [255, 0, 0]
    ammo = [255, 255, 0].
```

Create your own sounds

Customise your game and create your own voice instruction, commands and player updates. You can record these with a microphone or use a text-to-speech website (http://www.fromtexttospeech.com). Download the files and save in MP3 format to play them through the Pygame interface.



06 Add data to Sea2 list

Remember the empty list called Sea2 which you created in Step 4? In this step you'll populate the list. First create a for loop, to loop over another list called Sea (which is created in **Step 17**). Select a random number between 1 and 5, and depending on what that is, add an item to the list. (There are 64 items, one for each LED.) Use conditionals to determine which items are added. To ensure there is more sea than ships, it can be selected three times, resulting in a 60% chance that 'water' will be added. At the end of this step, your list, Sea2, will contain a total of 64 items consisting of a random combination of water, ships and ammo. When the program runs, each of these entries will be looked up in the colour codes you created in Step 5 and returned to the list. This list can then be used to plot the relevant LED colour.

```
  for i in Sea:
    item = random.randrange(1, 6)
    if item == 1:
        Sea2.append(ship)
    elif item == 2:
        Sea2.append(water)
    elif item == 3:
        Sea2.append(water)
    elif item == 4:
        Sea2.append(water)
    elif item == 5:
        Sea2.append(ammo)
```

07 Count ships

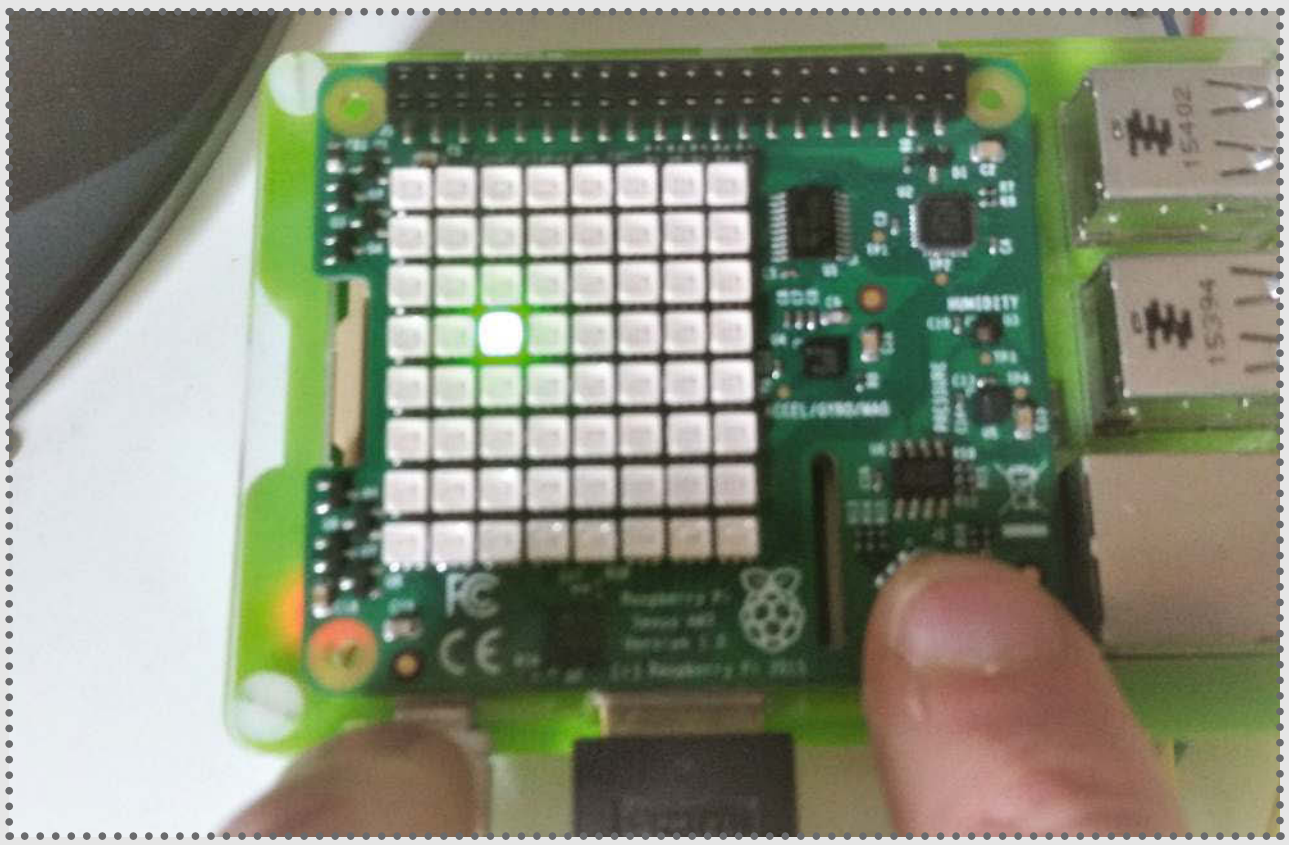
This step counts the number of ships that have been added to the Sea2 list. This is a random number and will change each time the game runs. Begin by creating a variable to store the number of ships. Set this to zero so

it resets for each new game. Then check each individual entry in the Sea2 list for the ship colour; in this example, (160, 200, 140). If the value for a ship is found, then increment the number of ships variable by one. This variable now stores the total number of ships for you to find and destroy.

```
  '''Counts how many ships there are in the
   sea'''
  '''Looks through Sea2 list and counts number
   of ships'''
  number_of_ships = 0
  for entry in Sea2:
    if entry == [160, 200, 140]:
    number_of_ships = number_of_ships + 1
```

08 Load torpedoes

Now that the game has calculated the number of enemy ships, inform the player by scrolling a message across the LED matrix. You can customise the colour by changing the text_colour value; this uses the same



RGB colour values as before. Initialise the number of torpedoes by creating a variable and adding five to the total number of ships. Adapt the number at the end to make the game easier or harder to play. (When testing, it's worth setting the count to five to save you having to deploy 30 torpedoes each time you test the game.)

09 Set up joystick – part 1

It's time to set up the joystick to control your boat and target a location to fire on. Your boat is represented by the green LED. We check that you still have torpedoes left; if so, then the game is still in play and you can move. Line four sets the LED to black and uses the x and y values from Step 3 to place your ship in the top-left corner of the LED matrix. Add the code to respond to a joystick movement; this checks for it being moved down and then increments y by a value of one so your ship appears to move down. Use an elif statement to respond to the joystick moving up and your ship LED moves up one position.

```
  while torpedos > 0:
    for event in pygame.event.get():
    if event.type == KEYDOWN:
        sense.set_pixel(x, y, 0, 0, 0)  # Black
    0,0,0 means OFF
        if event.key == K_DOWN and y < 7:
            y = y + 1
        elif event.key == K_UP and y > 0:

```

10 Set up joystick – part 2

Continue to add the code for the joystick being moved left or right. This follows a similar format as the y axis code but is applied to the x position. To fire a torpedo,

press Enter or push the joystick in; this subtracts one torpedo from your total. The indentation on this section is important: ensure that the elif statements are at the same level as the second if statement in the previous step.

```
  elif event.key == K_RIGHT and x < 7:
    x = x + 1
  elif event.key == K_LEFT and x > 0:
    x = x - 1
  elif event.key == K_RETURN:
    torpedos = torpedos - 1
    print ("You have", torpedos, "torpedos")
```

11 Check if you hit a ship

To check if you've hit an enemy ship, you need to compare your current position with the same position in the Sea2 list. If there is a ship, then you have hit it. For example, if you are at LED 43, this is compared with the data held at position 43 in the Sea2 list. The formula (y*8)+x is used to calculate your position as a number. We then check if the position equals a ship. If so, we update a different list called Sea with the 'hit' and then update and display the new LED colours, reading the from the Sea list.

```
  pygame.init()
  pygame.mixer.init()
  pygame.display.set_mode((200, 200))
  sense = SenseHat()
  #sense.low_light = True '''enable to turn
    down brightness'''
```

12 Adding an explosion sound

Red, Green and Blue colours

RGB is an additive colour model in which red, green and blue light is combined in various amounts to create a wide range of shades – over 16 million different variations. The lowest value for each of the three main colours is 0; if they're all 0, that results in black (the absence of colour). The top value for each RGB element is 255; if they're all set to 255, that's white.

In FileSilo, you'll find a folder of sound effects for your game or you can create and use your own. Ensure that the folder is named sounds or change the following code. First, we utilise the Pygame sound facility and load the MP3 file, 'an explosion for hitting the ship'. We then play the sound. Add a small delay of four seconds to allow the file to play; if your sound is longer, then extend the delay. Since you hit a ship, subtract one from the total number of remaining ships.

```
    ### sound ###
    pygame.mixer.music.load("sounds/impact.mp3")
    pygame.mixer.music.play()
    time.sleep(4)
    number_of_ships = number_of_ships - 1
```

13 Update the player

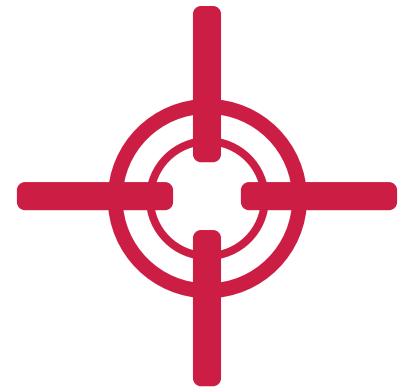
The player hit a ship; now to update them with the current number of remaining enemy ships. This is announced using a sound file. The number of ships remaining is then scrolled across the LED matrix. As before, you can set the colours by adjusting the RGB values. Finally, increment the score by one.

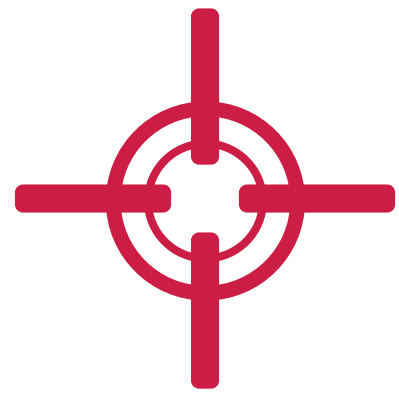
```
    pygame.mixer.music.load("sounds/shipsleft.
      mp3")
    pygame.mixer.music.play()
    sense.show_message(str(number_of_ships),
      text_colour=[255, 0, 0])
    score = score + 1
```

14 Hitting water, a miss

If you don't hit a boat, chances are your torpedo hit open water. An elif statement is used to check the position of

your ship against the same position in the Sea2 list. If it returns water, then update the Sea list. Turn the LED blue, then load the 'miss' MP3 file using the Pygame mixer. Play the sound, using the code pygame.mixer.music.play() and remember to leave a delay before moving onto the next program instruction.

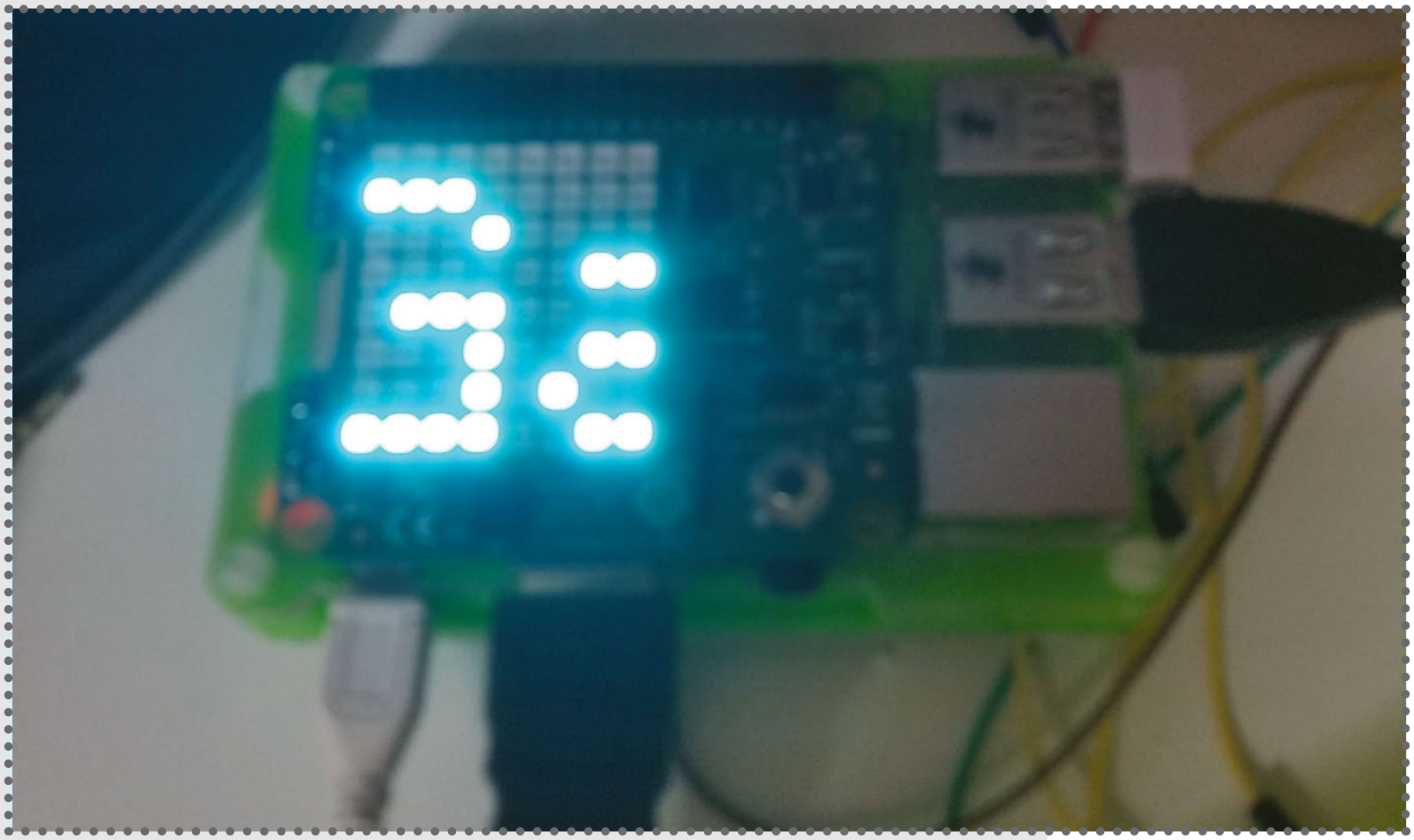




```
  elif Sea2[your_position] == water:
    print ("MISS")
    Sea[your_position] = water
    sense.set_pixels(Sea)
    ### sound ###
    pygame.mixer.music.load("sounds/miss.mp3")
    pygame.mixer.music.play()
    time.sleep(1)
```

15 Find additional ammo

During the game you may be lucky enough to find a



store of ammo, which will be added to your arsenal of torpedoes. The code for this follows the same structure as the previous **Steps 11** and **12**. First, you need to check if your position corresponds to 'ammo' and then update the Sea list. If so, increment the torpedo variable by one and then load and play the ammo MP3 file. Watch the indentation on these lines – see the full code listing for the correct layout.

```
  elif Sea2[your_position] == ammo:
    Sea[your_position] = ammo
    sense.set_pixels(Sea)
    torpedos = torpedos + 1
    pygame.mixer.music.load("sounds/ammo.
mp3")
    pygame.mixer.music.play()
    time.sleep(3)
```

16 Ending the game

Back in **Step 9** you wrote the code to control the joystick and move your ship around the LED matrix. The program only checks and compares your position to the Sea2 list if you press Enter. If not, it simply displays your ship LED. When you press Enter, the program checks your location for another ship, water or ammo, updating the Sea list and playing the related MP3 file. This cycle continues until the number of torpedoes equals zero, then the 'game over' MP3 file is loaded and played. A final 'game over' message is scrolled across the LEDs using the code sense.show_message().

```
  sense.set_pixel(x, y, 0, 255, 0) #colour of
    pixel for location target
  pygame.mixer.music.load("sounds/gameover_
    spoken.mp3")
```

```
    pygame.mixer.music.play()
    sense.show_message("Game Over", text_
      colour=[0, 255, 255], scroll_speed=0.07)
```

17 Testing the game

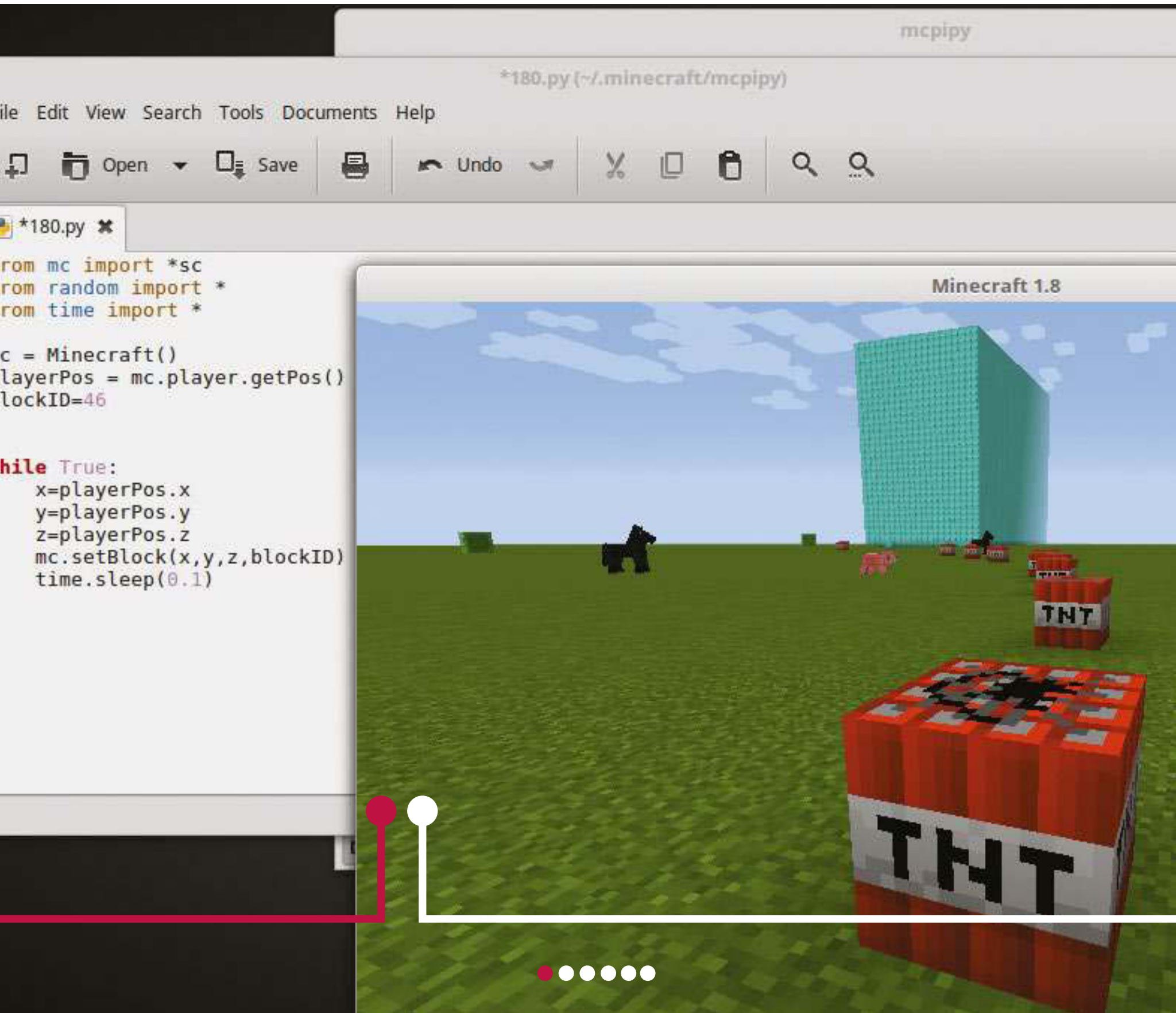
This concludes part one of the tutorial. You will probably want to test the code and play a basic version of your game. First, create a variable called S and assign a colour. Then create the Sea list referred to in Steps 11, 12, 14 and 15. This contains 64 entries, one for each LED, and is used to hide enemy ships and ammo. Write the colours to the LED matrix. Add a short time delay before running the main program. Save your game, pressing F5. After a short intro, you can move your ship around with the joystick; press Enter to torpedo a position. Good luck, captain!

```
    '''Creates a grid of 64 blue pixels, the
      sea'''
    Sea = [S, S, S, S, S, S, S, S, S, S, S, S, S,
      S, S, S, S, S, S, S, S, S, S, S, S, S, S, S,
      S, S, S, S, S, S, S, S, S, S, S, S, S, S, S,
      S, S, S, S, S, S, S, S, S, S, S, S, S, S, S,
      S, S, S, S, S, S]
    sense.set_pixels(Sea)
    time.sleep(1)
    main()
```

See you next issue for the second part of this project!

Control the player and environment in Minecraft

Continuing our series on Python coding in Minecraft, this issue we're looking at controlling the player character and the world around them



Hooking directly into Minecraft with Python gives us many advantages, but rather than building by hand, we can use programming techniques such as loops to simplify much of the building process. In this issue's tutorial, we will manipulate the player character's position and the blocks around them. We use simple lines of code to replicate blocks in rows and columns, to build entire rooms out of thin air. We also teleport Steve or Alex around your world at the drop of a hat. Combining these two functions enables us to have a little fun with TNT to create our own bomber-man.

As a prerequisite, we assume you've installed McPiFoMo, from our last two issues. McPiFoMo includes MCPiPy by 'fleap' and 'bluepillRabbit' of MCPiPy.com; the Raspberry Jam Mod, developed by Alexander Pruss; and Minecraft Turtle by Martin O'Hanlon of **www.stuffaboutcode.com.**

THE PROJECT ESSENTIALS

Minecraft
https://www.mojang.com/games
Python
https://www.python.org
McPiFoMo
http://rogerthat.co.uk/McPiFoMo.rar
Minecraft Turtle
http://bit.ly/MinecraftTurtle
Block IDs
http://bit.ly/minecraft_id_links

01 Moving the player character

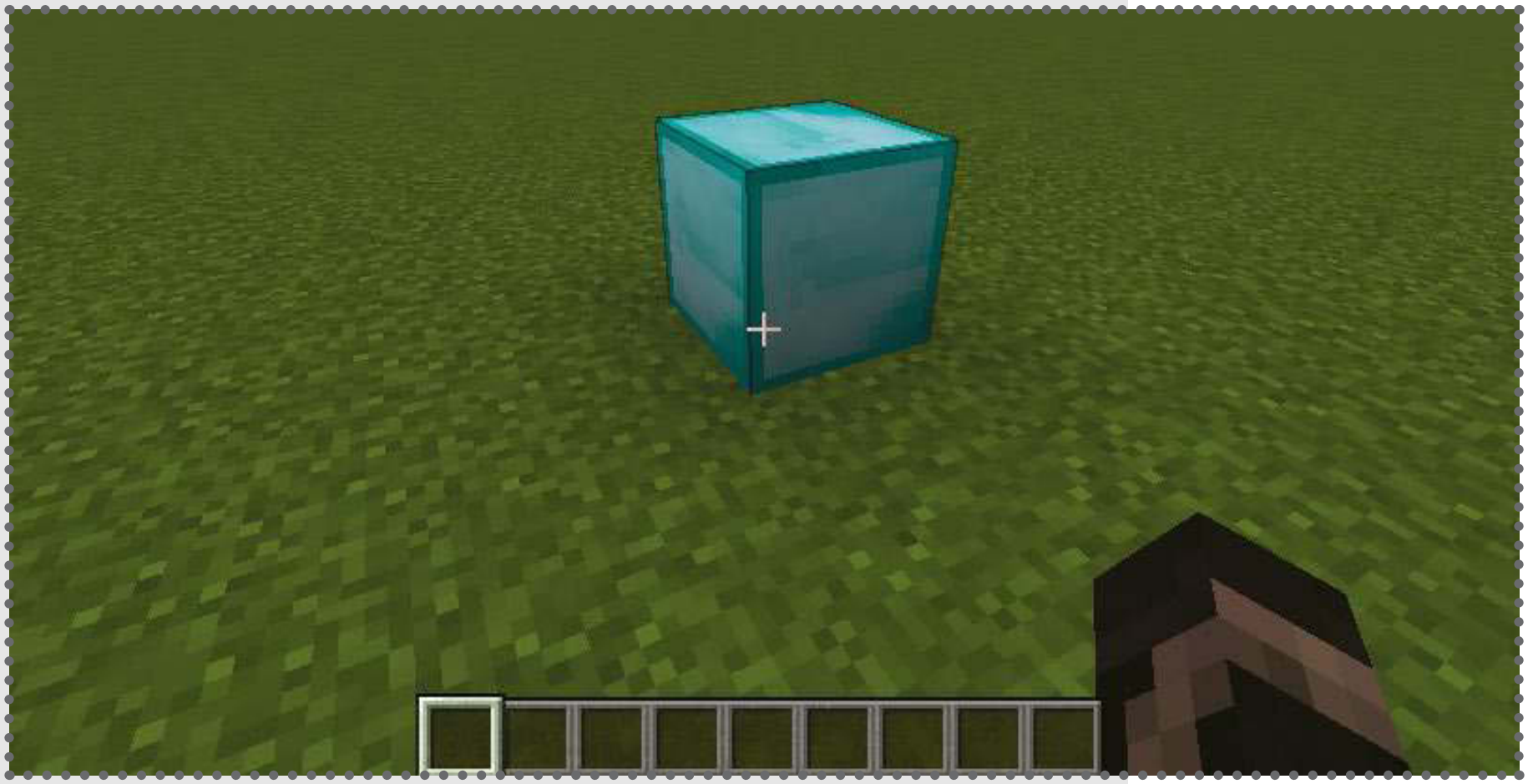
We can directly control our character's position using x,y,z coordinates. Create a new Python script and import mc.

```
from mcpi import minecraft
mc = minecraft.Minecraft.create()

mc.player.setTilePos(20, 20, 20)
```

Of course, we can replace these values with variables, to make things easier later on:

```
x=20
```



```
y=20
z=20
mc.player.setTilePos(x, y, z)
```

Now whenever we want to change the player's position, we simply alter the x,y,z variables.

02 Placing blocks

As well as moving the player character, we need the ability to place and move blocks around our world. To do this, we use the setBlock command:

```
mc.setBlock(30,30,30,57)
```

Again, variables make life easier, so in practice we'd probably arrange it like this:

```
x=30
y=30
```

```
  z=30
  blockID=57
  mc.setBlock(x, y, z,blockID)
```

The blockID for diamond is 57 – you can find more using the link in the What you'll need box.

03 Combining the two

If we assign the player position to a variable, we can use it as coordinates for placing blocks, so that we can create something around our player.

```
  playerPos = mc.player.getPos()
  x=playerPos.x
  y=playerPos.y
  z=playerPos.z
  mc.setBlock(x,y,z,blockID)
```

This will place a block exactly where our player is. You can then offset the coordinates accordingly: x+=1. Remember: x is East/West, z is North/South and y is Up/Down.

04 User input to define variables

With the addition of a typical input command, we can ask the user what type of block they'd like to create.

```
  blockID = input("Which blockID would you
like to use? ")
```

You can of course do the same for the y and z coordinates.

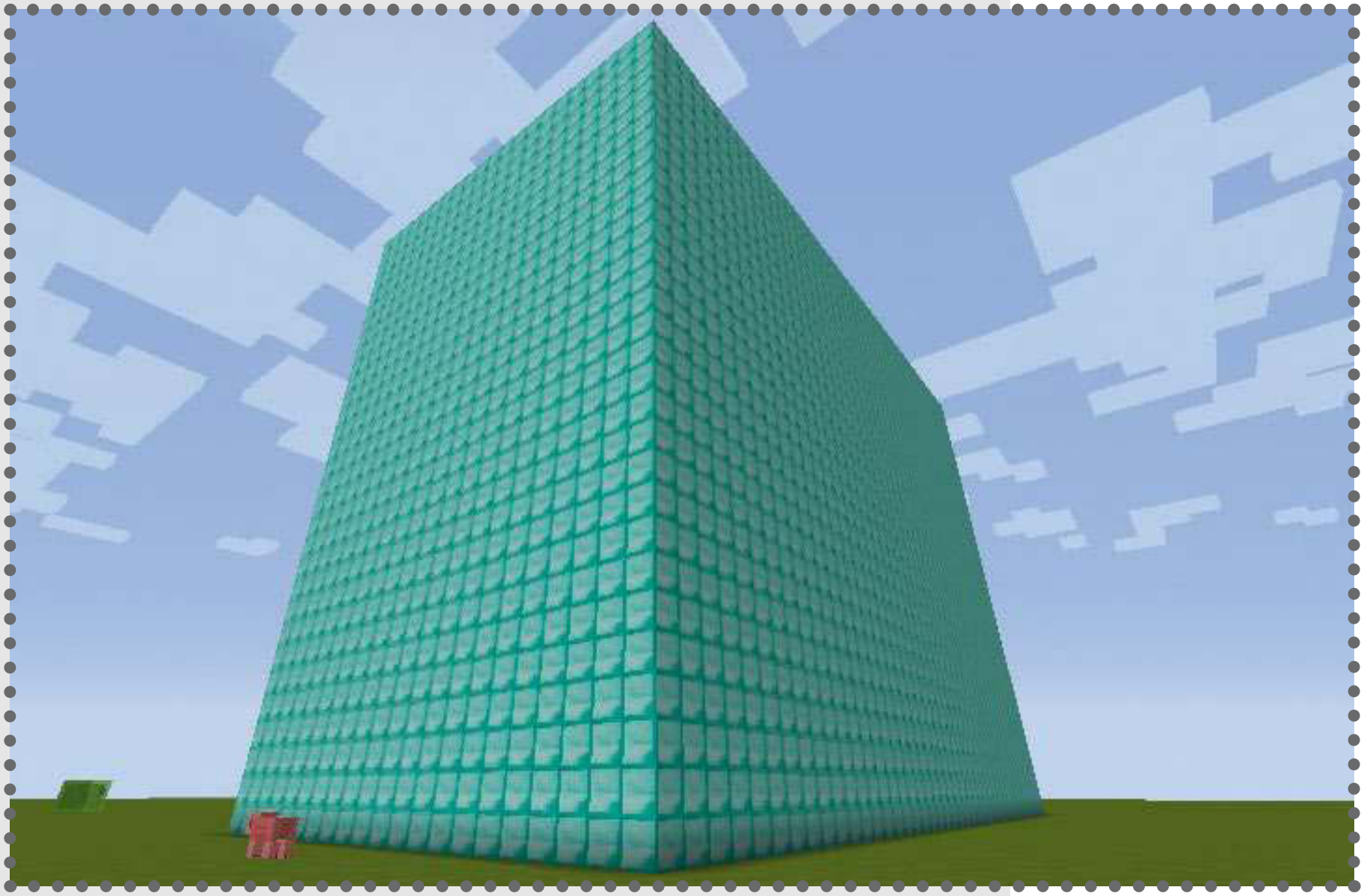
Take your time

Whether you're placing blocks or moving your player character around the world, you may want to space out your commands. Using Python's time.sleep command, we can create an artificial gap in our code. For instance, time.sleep(10) would pause the program for ten seconds before initiating the next line of code. This could be useful if you wanted a certain block to appear every x number of seconds, or if you fancy moving a player around at specific intervals.
In order to use time.sleep, you will need to import time at the top of your code.

05 Build in rows and columns

Placing blocks one at a time can slow down the process. There may be instances where we want to place entire rows and columns of blocks at once. After setting your playerPos and blockID (as in Step 3 and Step 4), we need to add variables for rows and columns.

```
  eastWest = 20
  northSouth = 30
  upDown =  40
  mc.setBlocks(x,y,z, x+eastWest,
y+northSouth, z+upDown, blockID)
  mc.setBlocks(x+1,y+1,z+1, x+eastWest -1,
y+northSouth -1, z+upDown -1, 0)
```

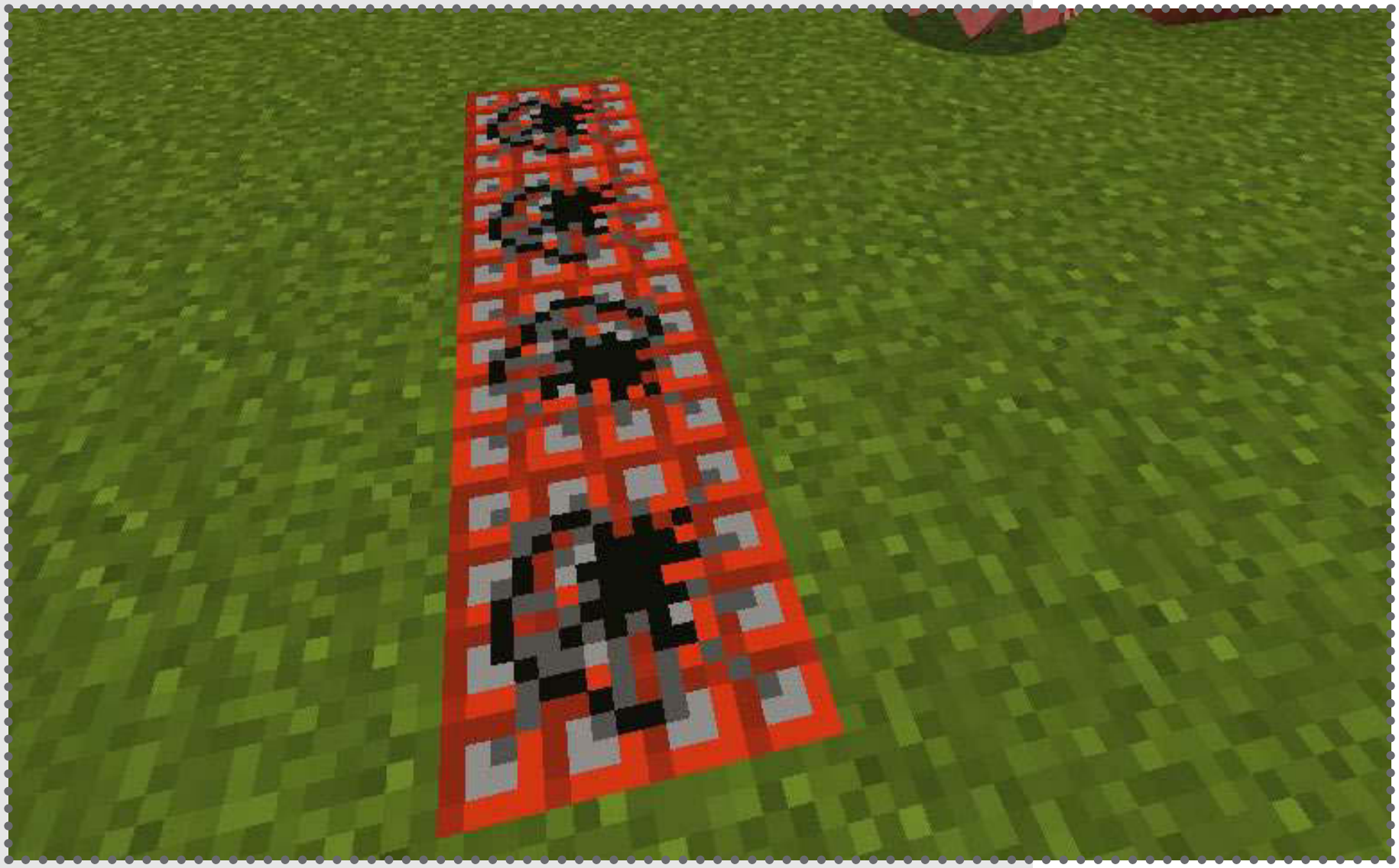


06 Explosion man

Now that we're familiar with playerPos and setBlock, we can combine the two to turn your player into a walking bomber-man. By moving our earlier code into a while loop, and with the addition of a simple Boolean, we can make our player drop bombs with every step:

```
while True:
    playerPos = mc.player.getPos()
    x=playerPos.x
    y=playerPos.y
    z=playerPos.z
    mc.setBlock(x,y,z,46)
    time.sleep(0.1)
```

Now it's time to have some explosive fun!



Recognise emotions with your Pi

How to teach your Raspberry Pi to recognise emotions, using either a web camera or the official Raspberry Pi Camera module

"In a previous issue, we taught the Raspberry Pi to recognise faces, using OpenCV. This month, we're taking this further to teach the Raspberry Pi how to recognise emotional content.

To do this, we will look at how to use the Google Cloud Vision API to analyse the images that the Raspberry Pi has taken. This API is free if you are just using it for personal projects, but keep in mind that if you use it for a large number of transactions, you will need to pay based on usage.

You start by getting an account set up with Google and creating a file with your authentication credentials. Read more about this in the official documentation at https://cloud.google.com/vision/docs/common/auth). Once you have a JSON file with your credentials, you can make it available to your code by executing the following command on the command line:

```
  export GOOGLE_APPLICATION_CREDENTIALS=/
path/to/credential/file.json
```

Since this is a web service, your Raspberry Pi will need to be on the internet to send the images to the Google Cloud and get the analysis results back. We will assume that you are taking the pictures with either the PiCamera or a USB webcam. You can take pictures with the PiCamera using code like the following:

```
import picamera
camera = picamera.PiCamera()
camera.resolution = (1600, 1200)
camera.sharpness = 100
camera.capture('image.jpg')
camera.close()
```

In either case, you should have the pictures of the faces you need on the filesystem of your Raspberry Pi, ready to be analysed.

When you have your images ready, the first step is to make sure that all of the required packages are installed. If you are using a Debian-based distribution (distro), such as Raspbian, you can install the required packages with the following commands:

```
sudo apt-get install python-oauth2client
sudo pip install --upgrade google-api
python-client
```

The second command is needed because there is no DEB package in the standard Raspbian package repository. You can now import the libraries that you will need to connect to the Google API and start

making requests. Input the following code:

```
  import base64
  from googleapiclient import discovery
  from oauth2client.client import
  GoogleCredentials
```

As you can see, we only need the discovery object, used to find the web service of interest, along with the GoogleCredentials object to handle the authentication to the Google service. Assuming that all of the Python modules installed correctly, the above code should import. The next step is to make the actual connection to the Google Vision services. The following code assumes that you have correctly set the environment variable earlier.

```
  credentials = GoogleCredentials.get
application_default()
   service = discovery.build('vision', 'v1',
credentials=credentials)
```

To use this service, you need to send your request in as a JSON data set. This set includes the image data containing the face that you want to analyse. Assuming that the face image is stored in the file image.jpg, the following code will load the image and send it to Google with open('image.jpg', 'rb') as image:

```
  image_content = base64.b64encode(image
read())
  service_request = service.images().
```

Why Python?

It's the official language of the Raspberry Pi. Read the docs at **python.org/doc.**

```
  annotate(body={ 'requests': [{'image':
{'content': image_content.decode('UTF-8')}
  'features': [{'type': 'FACE
DETECTION','maxResults': 10}]
  }]
  })
  response = service_request.execute()
```

As you can see, the image data needs to be base64 encoded in order to be sent to the Google vision service. Once you have the request generated, you can use the execute() method to send it off to the Google Cloud and get the results back. The Google Vision facial recognition returns a lot of details about the face that you sent in, and there is a list of the positions for all of the features of the face.

Along with physical information, the Google Vision service returns a list of potential emotional content, which is what interests us. There are four categories of emotional content returned, named joyLikelihood, sorrowLikelihood, angerLikelihood and surpriseLikelihood. The values of these categories are a percentage, given as a range between 0.0 and 1.0, that is somewhere between 0 per cent and 100 per cent. This specific percentage and what it represents may change as the model changes, so it is not that useful for an end user.

Luckily, there is a set of bucketised categories that are more intuitive. They are labelled **UNKNOWN, VERY_UNLIKELY, UNLIKELY, POSSIBLE, LIKELY** and **VERY_UNLIKELY**. You can then check to see how likely each emotion is in the given face. To simply see what

the values are, you can use the following code:

```
 anger = response['responses'][0]
['faceAnnotations'][0]['angerLikelihood']
  surprise = response['responses'][0]
['faceAnnotations'][0]['surpriseLikelihood']
  sorrow = response['responses'][0]
['faceAnnotations'][0]['sorrowLikelihood']
  joy = response['responses'][0]
['faceAnnotations'][0]['joyLikelihood']
  print("Happy: " + str(happy))
  print("Angry: " + str(anger))
  print("Surprise: " + str(surprise))
  print("Sorrow: " + str(sorrow))
```

You can now tailor the behaviour of your code based on these results. For example, if sorrow is very likely, you could ask the user what is wrong, as in the following example.

```
  if (str(sorrow) == 'VERY_LIKELY'):
  print('It looks like you are sad. Do you
want to talk about it?')
```

Since your Raspberry Pi is capable of dealing with physical objects through the GPIO pins, you could have it do something physical, like lighting particular LEDs or activating a servo.

The Google Vision service also recognises other items from the same call to the facial recognition service. Let's say you wanted to see if someone was wearing the hat of power. Luckily, the information returned from your service request includes the

"Make your Pi more responsive to people"

likelihood that the person in the image is wearing some headwear. Using this data, the following code could be used to unlock the doors to your clubhouse:

```
  joy = str(response['responses'][0]
['faceAnnotations'][0]['joyLikelihood'])
  headwear = str(response['responses'][0]
['faceAnnotations'][0]['headwearLikelihood'])
if (joy == 'VERY_LIKELY'):
  if (headwear == 'VERY_LIKELY'):
  unlock_system()
```

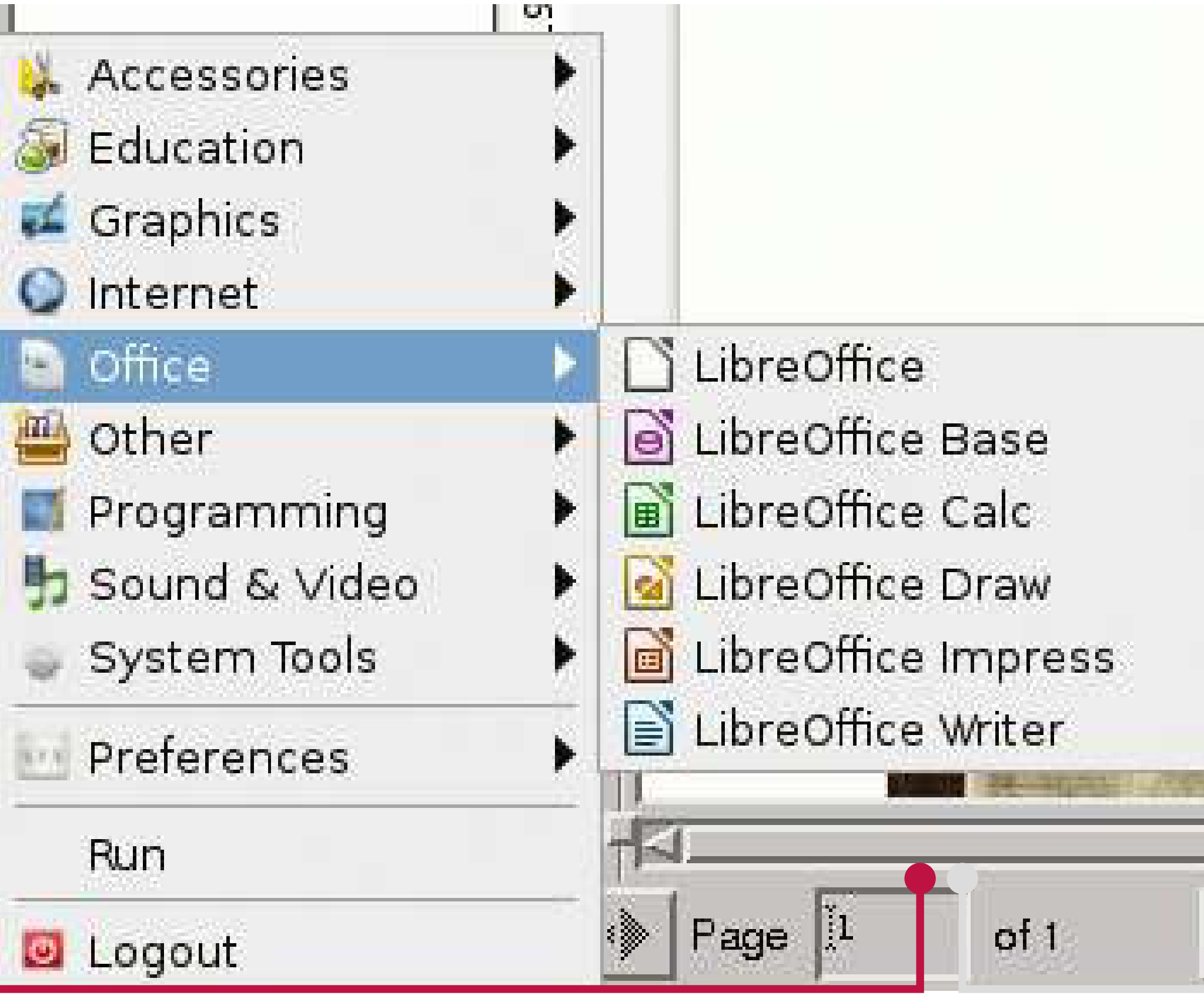
The vision service can be used for several other recognition tasks, such as detecting broad categories of objects within a given image. With the following code, you could send an image of your living room and get back whether your pet cat is in that image:

```
description = str(response['responses'][0]
['labelAnnotations'][0]['description'])
  if (description == 'cat')
  print('Hello kitty')
```

This code uses the same type of service request as above. Except that it is changed from **'FACE_DETECTION'** to **'LABEL_DETECTION'**. If you are analysing images of the outdoors, you could use the landmark detection method to see what natural features exist within a given image. It tries to identify specific landmarks and its geographical location, along with a confidence score. With the power of Google's AI behind you, you can make your Pi more responsive to the people it needs to interact with.

Turn your Pi into an office suite

How to add a full, free office suite to your Raspberry Pi



Common tasks when using any computer are word processing, spreadsheets and presentations. This is well within the scope of the Raspberry Pi. Raspbian comes bundled with a couple of text editors which are fine for making simple notes. However, as soon as you come to creating actual documents and want any kind of formatting or pictures inside them, these editors really aren't up to the task and so you'll want to move to something new.

The best way to get round this is to install another application – or, as is the case here, a full office suite! This is really easy on the Pi and the best option available is open source and completely free: LibreOffice.

Installing it couldn't be much easier – and it gives a lot of scope to what you're able to use your Raspberry Pi for. It's fully featured and so long as you're not creating huge documents with hundreds of images in, will be more than adequate for most of your needs.

To get started, you'll need a working internet connection for this tutorial, so make sure your LAN cable or wireless dongle is plugged into your Pi and functional, because we'll be using the built-in package manager.

Installation and set-up time can vary depending on your home set-up, but aim to set around 30 minutes aside and you won't be far wrong. The screen shouldn't stay still for too long, so you'll know it's doing something!

01 Install LibreOffice

The best thing about open source software is the fact that you can install it any time (so long as you have an internet connection or installation medium anyway) at your total convenience – without having to worry about

handing over payment details. To install the LibreOffice office suit by using the package manager, type the following
into Terminal:

```
sudo apt-get install libreoffice
```

Now type in your password. Press 'Y' and Enter when prompted to install other dependency packages. Sit back, relax and enjoy. Note that when asked to type your password for sudo, you won't actually see it being typed on screen.

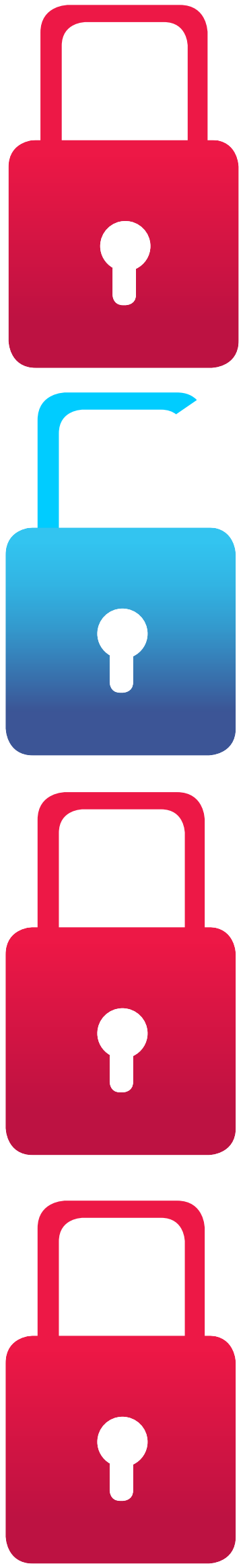
02 Explore the suite

When you've installed LibreOffice, you can admire your new applications by hitting the system menu. You'll see a new 'Office' category that's shown up, underneath which you'll see the following applications have been installed. That's right, it's not just for word processing!

LibreOffice Base – A database management application.
LibreOffice Calc – For looking after spreadsheets.
LibreOffice Draw – To make flowchart/visualisations.
LibreOffice Impress – For presentations.
LibreOffice Writer – The word processor.

These applications are perfectly able to open a majority of documents created in their expensive Microsoft Office counterparts and are almost as feature rich. But we'll now explore a few of the Writer features in more detail.

03 Add some style



We're focusing on some of the word-processing features on the Pi specifically here, so go on ahead and open up LibreOffice Writer. When you're writing documents, it helps to put some formatting in. It helps with reading the document, and it looks prettier too. When you start writing, to the left of the font drop-down menu there's another drop-down that currently reads 'Default'. Select some text, then use this drop-down to change the current applied style. If you don't like the defaults, select 'More' from the drop-down, or press F11. This will allow you to customise the styles that exist already, switch an existing style to properties of the selected text, or add new ones.

04 Add an index

One of the great things about using formatting is that it allows you to create a contents page for your work really easily. So long as you use consistent headings for the different levels of your document, it's a quick and easy way to give a guide to what's in your document. Another advantage is that if afterwards you export the file to PDF, you'll end up with each heading in the contents hotlinking directly to the correct section within your document.

To create your index, create some space at the top of your document, and then insert your index by using the menus to go to 'Insert Indexes and Tables>Indexes and Tables'. Review the selected options and click OK when you're happy. It'll insert to the section of the document that your cursor is on.

05 What about images?

There's often a requirement to put images in a document, and naturally this is something we can do relatively easily.

There are a couple of ways of doing this – the easiest one by far is to drag an image in from the file manager directly into the document. Open the folder in file manager. Say it's on your desktop, simply drag the file into LibreOffice Writer and you will see the icon change; upon letting go, your image will appear wherever you drop it.

Alternatively, you can use the Insert>Picture> From File option. This gives you an open dialog that allows you to do exactly the same thing.

06 Export as a PDF

The PDF format is a very common way of exchanging documents because you don't really need to worry about the recipient having a specific application to look at them in as most modern operating systems will have some sort of PDF viewer built in. It's also a great way of sharing documents without worrying that someone will accidentally make changes to a documents. It's even possible to set passwords on PDFs to stop purposeful modifications to them.

In LibreOffice, you can save your document as a PDF using the File>Export to PDF menu option. Choose your desired options, click Export and save it in the appropriate place.

07 Add comments

When writing a long document you'll often end up thinking of things that you should really check up on or add to the document later. When you think of these things, you can make notes about them – but physical notes aren't always that ideal. There's a little-known function in most word processors now for 'Comments' – even more useful when there are multiple people moving through the document. Comments can be added at the current point in the document with Ctrl+Alt+C or by using the menu option Insert> Comment. If you want to get rid of the extra 'Comments' pane, you can use the View menu to toggle them: View>Comments.

08 Add links

Say you've used outside resources in a document that you want to reference, or maybe you don't directly need to include the information though it's useful – it might be handy to have a way of getting back to that document later on. You could have an Appendix at the end of your document that links to these other resources. You can hyperlink to these easily using the Hyperlink toolbar button or the same from the Insert>Hyperlink menu. Select your text, then click it and you'll be presented with a dialog to link it the web. It's important to note that direct document links use full paths. If it's being sent to someone else, avoid using these as they'll usually break. Ideally use it only a personal reference guide.

Next issue

Get inspired **Expert advice** **Easy-to-follow guides**

Build your own HI-FI

Get this issue's source code at:

www.linuxuser.co.uk/raspicode